uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática

2017 - 2018

Trabajo Fin de Grado

"Desarrollo de Interfaz para Visualización de Algoritmos de Asistencia a la Conducción"

Diego Martínez González

Tutor/es

Jorge Beltrán de la Cita

Leganés, junio de 2018

RESUMEN

En la presente memoria se expone el estado del arte actual relacionado con el vehículo autónomo y los sistemas y elementos relacionados con el mismo. Además, se exponen los objetivos buscados, los recursos y entornos utilizados para llevar a cabo el proyecto, así como las aplicaciones creadas para la consecución de dichos objetivos.

Para ello, se expone el desarrollo de una interfaz que permita visualizar el entorno captado por un vehículo autónomo a través del procesado de imágenes y nubes de puntos. Por otro lado, también se explican los métodos utilizados y el desarrollo de una aplicación con sistema operativo (SO) Android, a través de la cual se pueden visualizar los entornos mapeados de nubes de puntos y las imágenes captadas por los diferentes sensores y cámaras del vehículo mencionado. Además, se muestran los resultados obtenidos de los sistemas implementados.

Por último, se da a conocer el marco legal en el cual queda englobado este proyecto, el entorno socioeconómico actual y los trabajos futuros y de mejora que se pueden aplicar sobre las aplicaciones desarrolladas, así como el presupuesto necesario para su desarrollo.

Palabras clave:

ROS (Sistema Operativo Robótico), ADAS (Sitema avanzado de Asistencia a la Conducción), Vehículo Autónomo, Sensores, Nubes de Puntos (PCD), Visión por Computador.

ABSTRACT

The present project report is an exposition of the current state-of-the-art related to Autonomous Vehicle and the elements and parts associated to it. Moreover, the marked objectives, and the resources and environments used to carry out the project are set out, as well as the applications created to achieve these goals.

For this, the development of an interface that allows visualize the captured environment by an Autonomous Vehicle by processing images and point clouds is set forth. Furthermore, there´s an explanation of the methods used and development of an Android (OS) App based on, through which you can visualize the mapped environments of point clouds and images captured by the different cameras and sensors mounted in the vehicle. In addition, the results obtained from are displayed the implemented systems.

To sum up, the legal framework in which this project is included, the existing socio-economic environment, and the future and improvement works, that can being applied to the development applications, are made public.

Keywords:

ROS (Robot Operating System), ADAS (Advanced Driver-Assistance System), Autonomous Vehicle, Sensors, Pointclouds (PCD), Computer Vision.

AGRADECIMIENTOS

A tutores y profesores que me han ayudado, a mis compañeros de guerra en la Universidad, algunos de ellos amigos, y a mi hermano de vida paralela.

A mis padres, Eduardo y Rosa, mi abuela Candelas, mis tíos y a mi prima Bárbara, por su apoyo y confianza incondicional en mí, y simplemente por estar ahí. También a ti, tía, que no dudaste de mí, y hoy seguro sabes que he llegado al fin.

Y por supuesto, a ti Inés, mi mayor apoyo, mi mayor desahogo y mi principal motivación para levantarme y luchar por esta oportunidad.

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN 1
1.1. Motivación 1
1.2. Objetivo del estudio 2
2. ESTADO DEL ARTE 3
2.1.Ingeniería de Sistemas 3
2.1.1. Capacidades de la ingeniería de Sistemas 3
2.1.2. Problemática actual y avances 4
2.2. El vehículo autónomo 4
2.2.1. Introducción 4
2.2.2. Historia del vehículo autónomo 5
2.2.3. Sistemas de ayuda a la conducción (*ADAS*) 7
2.2.4. Principales bases de datos 8
2.2.5. Tipos de componentes del vehículo autónomo 10
2.2.6. Sensores 10
2.2.7. El vehículo autónomo IVVY 2.0 13
2.2.8. Sistemas de visualización 14
2.2.9. Android Auto 15
2.2.10. Plataforma *Apple* para la conducción autónoma 16
3. RECURSOS 17
3.1. Sistema operativo *Ubuntu Linux* 17
3.1.1. Introducción 17
3.1.2. Instalación del SO 17
3.2. ROS (*Robot Operating System*) 18
3.2.1. Introducción a ROS 18
3.2.2. Instalación del entorno 18
3.2.3. Descripción de elementos 19
3.3. *Qt Creator* 20
3.3.1. Instalación 21
3.4.*Android Studio* 21
3.4.1. Introducción 21
3.4.2. Instalación del entorno 21
3.4.3. Descripción de elementos 22
3.5.*Git* 23
3.6. Librerías *OpenGL* 24

4. DESCRIPCIÓN DEL PROYECTO 26
4.1. Visualizador de nubes de puntos 26
4.2. Aplicación Android: *LSI Play Remote* 29
4.2.1. Introducción 29
4.2.2. Diagrama de bloques del modelo propuesto 29
4.2.3. Desarrollo y funcionamiento de la aplicación 30
4.2.4. Otros modelos implementados 35
4.2.4.1. Modelo de actividades múltiples 35
4.2.4.2. Modelo de actividades múltiples con nodo encapsulado 36
4.2.5. Ventajas e inconvenientes del modelo propuesto 37
5. RESULTADOS EXPERIMENTALES 38
5.1. Demostración de funcionamiento del visualizador 38
5.2. Demostración de funcionamiento de la *app LSI Play Remote* 42
6. TRABAJOS FUTUROS 46
6.1. Mejoras sobre el visualizador 46
6.2. Posibles mejoras sobre la aplicación de móvil 46
7. MARCO REGULADOR 48
7.1. Legislación vigente en el vehículo autónomo 48
7.2. Legislación en el desarrollo Software 50
8. ENTORNO SOCIOECONÓMICO 51
8.1. Impacto social 51
8.2. Principales mercados y cambios 52
9. PRESUPUESTO 53
10. CONCLUSIONES 55
11. BIBLIOGRAFÍA 57

ÍNDICE DE FIGURAS

Fig. 2.1. Vehículo Autónomo de Uber sin conductor 3
Fig. 2.2. Vehículo Autónomo de Google: *"Waymo"* 5
Fig. 2.3. Vehículo Autónomo de Tesla 6
Fig. 2.4. Vehículo Autónomo de Tesla: Interior 6
Fig. 2.5. Stanley, campeón de la edición de 2005 7
Fig. 2.6. Detector de ángulo muerto 1 7
Fig. 2.7. Detector de ángulo muerto 2 7
Fig. 2.8. Detector de señales 8
Fig. 2.9. Mapeo con *optical* flow 9
Fig. 2.10. Ejemplo de segmentación y reconocimiento de objetos 9
Fig. 2.11. Ejemplo de mapeo con *KITTI Vision Benchmark* 9
Fig. 2.12. Calibración de cámara simple 11
Fig. 2.13. Calibración de cámara omnidireccional 11
Fig. 2.14. Sensores de ultrasonidos 12
Fig. 2.15. Radar utilizado en aviones y barcos 12
Fig. 2.16. Detección de un sensor lídar 13
Fig. 2.17. Vehículos Autónomos de la Universidad Carlos III de Madrid 13
Fig. 2.18. Tecnología de visualización de Tesla basada en Nvidia 15
Fig. 2.19. Tecnología de Porsche basada en Nvidia 15
Fig. 2.20. Sistema Android Auto 16
Fig. 3.1. Estructura del funcionamiento básico de ROS 19
Fig. 3.2. Apartado de diseño de Android Studio 22
Fig. 3.3. Diseño de la aplicación a través de código 22
Fig. 3.4. Jerarquía de *Layouts* 23
Fig. 3.5. Jerarquía *src* 23
Fig. 3.6. Depurador de *Android Studio* 23
Fig. 3.7. Arquitectura de *Git* 24
Fig. 3.8. Figuras con *OpenGL* 1 25
Fig. 3.9. Figuras con *OpenGL* 2 25
Fig. 3.10. Ejemplo de aplicación con *OpenGL* 25
Fig. 4.1. Ejemplo de nube de puntos: Taza. 26
Fig. 4.2. Espacio de trabajo del visualizador 27
Fig. 4.3. Diagrama del flujo de datos del *3D Viewer* 27
Fig. 4.4. Diagrama de funcionamiento de la *LSI Play Remote App* 29
Fig. 4.5. Diagrama del flujo de topics desde el PC a la aplicación 32

Fig. 4.6. Diagrama del flujo de imágenes recibidas en la aplicación 33
Fig. 4.7. Diagrama explicativo de la recopilación de datos *PCL* 34
Fig. 4.8. Flujo de datos para la impresión de las nubes de puntos 34
Fig. 5.1. Estructura de compilación del visualizador 38
Fig. 5.2. Visualizador de *Point Clouds*. Tiempo de ejecución: 3.83s 39
Fig. 5.3. Visualizador de *Point Clouds*. Tiempo de ejecución: 6.20s 39
Fig. 5.4. Visualizador de *Point Clouds*. Tiempo de ejecución: 6.41s 40
Fig. 5.5. Gesto de zoom.....40
Fig. 5.6. Vista del visualizador con diferentes escalas de zoom 41
Fig. 5.7. Ejes que guían las posibles rotaciones explicadas 41
Fig. 5.8. Vista del visualizador desplazada 41
Fig. 5.9. Vista de la clase *MasterChooser*.....42
Fig. 5.10. Intento de arranque con dirección IP errónea.....42
Fig. 5.11. Arranque de la *app* con dirección IP correcta.....42
Fig. 5.12. Vista del listado: Visualización de imágenes a color y en escala de grises 43
Fig. 5.13. Visualización de *topic Image* 1 43
Fig. 5.14. Visualización de *topic Image* 2 43
Fig. 5.15. Visualización de *topic Image* 3 43
Fig. 5.16. Visualización de *topic Image* 4 43
Fig. 5.17. Visualización de *topic* PCL 1.....44
Fig. 5.18. Visualización de *topic* PCL 2.....44
Fig. 5.19. Visualización de *topic* PCL 3.....44
Fig. 5.20. Visualización de *topic* PCL con *Rviz* 44
Fig. 5.21. Ejemplo de salida de la aplicación 45

ÍNDICE DE TABLAS

Tabla 1.1. Personas accidentadas en España en 2015 1
Tabla 1.2. Personas accidentadas en España en 2016 1
Tabla 9.1. Tabla de amortización simplificada de la Agencia Tributaria.....53
Tabla 9.2. Costes de materiales generados.....54
Tabla 9.3. Costes de recursos humanos.....54
Tabla 9.4. Presupuesto final 54

ÍNDICE DE ABREVIATURAS

ABS: *Antiblockiersystem*
ACAHSRA: *Advanced Cruise-Assist Highway System Research Association*
ADAS: *Advanced Driver Assistance Systems*
CAN: *Controller Area Network*
DARPA: *American Defense Advanced Research Projects Agency*
DGT: Dirección General de Tráfico
GUI: *Graphic User Interfaces*
GNSS: *Global Navigation Satellite System*
GPL: *General Public License*
GPS: *Global Positioning System*
iCab: *intelligent Campus Automobile*
IDE: *Integrated* Development *Environment*
IMU: *Inertial Measurement Unit*
IP: *Internet Protocol*
IVVI: *Intelligent Vehicle based on Visual Information*
KITTI: *Karlsruhe Institute of Technology & Toyota Technological Institute*
LSI: Laboratorio de Sistemas Inteligentes
NAHSC: *National Automated Highway System Consortium*
OMS: Organización Mundial de la Salud
OpenGL: *Open Graphics Library*
PC: *Personal Computer*
PCL: *Point Cloud*
PIB: Producto Interior Bruto
QML: *Qt Meta Languages*
RAM: *Random Access Memory* RGB: *Red Green Blue*
ROS: *Robot Operating System*
RRHH: Recursos Humanos
RTLS: *Real Time Locating System*
RTVE: Radio Televisión Española
SO/OS: Sistema operativo / *Operative System*
USB: *Universal Serial Bus*

INTRODUCCIÓN

1.1. Motivación

El transporte motorizado, y, por tanto, el automóvil, se han convertido en la actualidad en un elemento indispensable en el desarrollo de las vidas cotidianas de la mayoría de familias. Tanto es así, que en el año 2014 ya se contaba con un parque automovilístico mundial en torno a los 1.200 millones de vehículos en circulación, un dato que sigue incrementando hasta la fecha [1].

Por otro lado, el avance en tecnología en la sociedad de la información también sigue una línea exponencial. Las investigaciones e innovaciones tecnológicas en todos los ámbitos y sectores son de gran importancia. Las empresas destinan a la investigación gran parte de sus presupuestos, sabedores de que estos avances tecnológicos son los que marcan la diferencia en la sociedad actual. Un estudio recientemente publicado por la cadena RTVE (Radio Televisión Española) estima que en este año 2018 se destinarán más de 6.300 millones de euros para la investigación, aumentando en más de un 5% los presupuestos estimados para el año pasado [2].

Un estudio realizado por *Ipsos* y *The Boston Consulting Group* estima que los españoles pasan más de 9 horas a la semana desplazándose en este medio de transporte [3]. Por tanto, la seguridad y comodidad dentro de los vehículos se han convertido en elementos fundamentales para la conducción.

Además, según los últimos datos proporcionados por la DGT (Dirección General de Tráfico) de los años 2015 y 2016, representados en las siguientes tablas, se puede comprobar que el número de personas implicadas en accidentes en España ha aumentado en más de 15.000 unidades [4].

TABLA 1.1.
PERSONAS ACCIDENTADAS EN ESPAÑA EN EL AÑO 2015

Total				
ACCIDENTES CON VÍCTIMAS	**ACCIDENTES MORTALES A 30 DÍAS**	**FALLECIDOS**	**HERIDOS HOSPITALIZADOS**	**HERIDOS NO HOSPITALIZADOS**
97.756	1.559	1.689	9.495	124.960

"*Extracto de Tablas Generales de 2015*". *(DGT, 2015)*

TABLA 1.2.
PERSONAS ACCIDENTADAS EN ESPAÑA EN EL AÑO 2016

Total				
ACCIDENTES CON VICTIMAS	**ACCIDENTES MORTALES**	**FALLECIDOS**	**HERIDOS HOSPITALIZADOS**	**HERIDOS NO HOSPITALIZADOS**
102.362	**1.663**	**1.810**	**9.755**	**130.635**

"Extracto de Tablas Generales de 2016". (DGT, 2016)

Por tanto, con el fin de mejorar la seguridad y comodidad de las personas, y a través de la tecnología, en los últimos años se han empezado a desarrollar sistemas que mejoren la calidad de la conducción. Entre estos sistemas, se encuentran los llamados Sistemas de Ayuda a la Conducción. Algunos de ellos son de mayor antigüedad como ocurre con el cinturón de seguridad, el *airbag* o el sistema de frenado ABS (*Antiblockiersystem)*.

En los últimos años se han empezado a desarrollar sistemas capaces de tomar decisiones en el vehículo sin la actuación activa del conductor, como los sistemas de frenado automático, sistemas de detección de carril o el *Park Assist*, un sistema capaz de aparcar el vehículo de manera autónoma. Todos estos sistemas necesitan de la implementación de sensores en los vehículos: sensores de posición, de proximidad, cámaras o sensores láser entre otros.

Por tanto, la importancia que tienen estos sistemas en la actualidad y el margen de mejora que tienen, dada su corta edad de desarrollo, son los principales motivos para la realización de este proyecto.

1.2. Objetivo del estudio

La Universidad Carlos III de Madrid y el departamento de Sistemas Inteligentes se encuentran inmersos en la investigación y desarrollo de los sistemas de ayuda a la conducción, los cuales están siendo implementados entre otros, en el vehículo IVVI 2.0 (*Intelligent Vehicle based on Visual Information*).

Para el correcto funcionamiento de los sistemas que se están implementando en ese vehículo, es importante el análisis de los datos obtenidos por los sistemas de detección instalados en el coche: sensores, cámaras…

Por lo tanto, se realiza la implementación de un visualizador de nubes de puntos para cumplir los siguientes objetivos:

1. Facilitar y mejorar la calidad del trabajo realizado por el Laboratorio de Sistemas Inteligentes.
2. Análisis y comprensión de los resultados obtenidos de los algoritmos implementados y del entorno en estudio.
3. Mejora de la confianza de los pasajeros respecto al vehículo a través de información visual.

Por otro lado, se crea una aplicación con sistema operativo *Android* para cumplir con los siguientes objetivos:

4. Integración de la aplicación con ROS, obteniendo un método de visualización que no requiere de implementación de entradas y salidas específicas, con lo que será muy útil de implementar en cualquier robot controlado a través de ROS.
5. Disminución de tiempos no productivos (desplazamientos laboratorio – taller).
6. Facilitar la visualización de algoritmos en dispositivos portátiles como teléfonos móviles o *tablets*.

2. ESTADO DEL ARTE

En este apartado se va a explicar la evolución hasta la actualidad de la ingeniería de sistemas, los sistemas de percepción y el uso que se está dando a dichos sistemas para la seguridad en automóviles. También se explicarán los principales entornos que se están utilizando para la consecución de resultados óptimos.

2.1. Ingeniería de Sistemas

La ingeniería de sistemas fue creada en el año 1943, surgiendo de la unión entre la ingeniería de conmutación y la ingeniería de transmisión. A pesar de trabajar como sistemas anteriormente, su reconocimiento como entidad organizativa generó un aumento de los recursos, facilitando así su desarrollo como comunidad [5].

A medida que fueron avanzando los años, el trabajo en el sector se fue incrementando gracias a las mejoras y avances que se obtenían a nivel informático. Aún, a principios del siglo XXI, los vehículos autónomos se consideraban un proyecto a largo plazo. Sin embargo, en este año 2018 se puede hablar ya de estos vehículos como una realidad, como se puede observar en la siguiente figura.

Fig. 2.1. Vehículo Autónomo de Uber sin conductor. *(El diario del Cibao, 2018)*

El avance tecnológico que se da en la actualidad basado en los propios avances informáticos, está permitiendo que el estudio en estos sistemas sea un campo en continuo desarrollo.

2.1.1. Capacidades de la Ingeniería de Sistemas

La ingeniería de sistemas está definida como el conjunto de ciencias matemáticas, físicas e informáticas, que, en consonancia con la electrónica y automática, es capaz de desarrollar sistemas que utilicen económicamente materiales tecnológicos para el beneficio de la humanidad, según explica la publicación de la web *Wikipedia*.

Con la unión de esas ciencias, es posible obtener:

- Transformación de una necesidad de operación en una descripción de parámetros de rendimiento del sistema y una configuración del mismo a través de procesos de interacción, diseño, pruebas…etc.

- Integración de parámetros técnicos y funcionales relacionados para asegurar la compatibilidad de todas las interfaces utilizadas en un programa con el fin de optimizar la definición de un sistema final.
- Integración de factores de fiabilidad, seguridad, humanos, y otros en el esfuerzo general con el fin de optimizar costes, obteniendo un buen rendimiento basado en la planificación.

2.1.2. Problemática actual y avances

El problema actual que se encuentra en el desarrollo de la ingeniería de sistemas, es la limitación de desarrollo. Esta limitación está impuesta por las marcas, de manera que estandarizan los sistemas y privan de libertad al desarrollador.

La solución se encuentra en el Software libre, que consiste en la utilización de sistemas que permiten trabajar libremente a través de estándares menos específicos que aportan mayor amplitud.

Por otro lado, los principales avances obtenidos de la ingeniería de sistemas vienen de la interacción con otras ramas, como lo son la inteligencia artificial y los sistemas de percepción, los cuales, son en la actualidad, dos de los campos en mayor expansión.

Entre sus aplicaciones se pueden encontrar:

- Lingüística computacional
- Minería de datos (*Datamining*
- Aplicaciones industriales
- Aplicaciones médicas
- Aplicaciones de robótica
- Sistemas de apoyo a la decisión
- Análisis de sistemas dinámicos

2.2. El vehículo autónomo

En este subcapítulo se procede a explicar la actualidad del vehículo autónomo, las características que lo definen y se expondrán temas importantes que hablen de la implementación de sus sensores y actuadores.

2.2.1. Introducción

El vehículo autónomo está definido como aquel “automóvil capaz de imitar las capacidades humanas de manejo y control” y como “una máquina capaz de percibir el medio que le rodea y navegar en consecuencia” tal y como explican Alexander Vicente Medina y Luis Alberto Galarza en su artículo [6].

Por tanto, se puede decir que el vehículo autónomo es aquel en el que no es necesaria la participación activa del conductor.

Estos vehículos perciben el entorno mediante el uso de técnicas complejas, sistemas de posicionamiento global y la utilización de la visión por computador. Uno de los ejemplos es el vehículo de *Google* mostrado en la figura 2.2.

Fig. 2.2. Vehículo autónomo de Google: *"Waymo". (Diario Expansión, 2017)*

Los sistemas avanzados de control utilizan la información para obtener la ruta hacia el destino teniendo en cuenta obstáculos, tales como atascos, cortes por obras o accidentes, distancia al punto exigido o el tiempo de traslado.

Estos vehículos son capaces de recorrer carreteras programadas previamente, las cuales tienen una reproducción cartográfica propia. Por tanto, si en una ruta se utiliza una carretera no recogida anteriormente por el sistema, el vehículo no podría avanzar de manera lógica y segura.

2.2.2. Historia del vehículo autónomo

La historia del vehículo autónomo es una historia reciente, pues sus primeros pasos se dieron a finales del siglo XX. Concretamente, en los años 80 se obtenían las primeras demostraciones satisfactorias.

Uno de los ejemplos conocidos es el proyecto europeo llamado *"PROMETHEUS"*, que involucró a trece fabricantes, y varios gobiernos y universidades. Bajo este proyecto, cabe destacar a Ernst Dickmanns, ingeniero que, apoyado por la inversión de las principales marcas en el proyecto, implementó un sistema de detección de líneas de la carretera en una furgoneta Mercedes-Benz en el año 1986 [7].

Por otro lado, en el mismo año 1986, en Estados Unidos nació el *"Navlab Thorpe 1"*, presentado por la *Carnegie Mellon University* y ligado a la organización *NAHSC (National Automated Highway System Consortium)*. Dicho vehículo circulaba a 2 km/h durante sus primeras simulaciones hasta llegar a su máxima velocidad unos años después, los 30 km/h. Llevaba seis *racks* en su interior con ordenadores, tres estaciones de trabajo, un receptor GPS y una pantalla de visualización, todo controlado por un "superordenador", según comenta Nacho Palou en su artículo "NavLab, un coche autónomo en 1986" [8].

A su vez, en Japón se formó la *AHSRA (Advanced Cruise-Assist Highway System Research Association)* con el fin de llevar a cabo sus proyectos de conducción autónoma.

Desde entonces, ha habido un gran progreso en el ámbito del vehículo autónomo. La realidad es que la conducción inteligente comienza a formar parte del mundo actual. A pesar de estos avances obtenidos, la conducción inteligente está aún a décadas de ser algo posible como explican Joel Janai y sus compañeros en el artículo publicado "*Computer Vision for Autonomous Vehicles*" [9]. Explican que esta afirmación está basada en dos motivos:

- En primer lugar, porque la conducción autónoma debe ser dirigida por sistemas inteligentes capaces de operar en entornos dinámicos de manera rápida en situaciones impredecibles.
- Y, en segundo lugar, porque la toma de decisiones requiere una gran precisión en sistemas de percepción, tema que aún no se puede considerar como tal, ya que los sistemas de visión por computador actuales generan errores en un porcentaje demasiado alto como para ser aceptable en este ámbito.

Uno de los fabricantes más involucrados en este ámbito es *Tesla*, fabricante del cual se muestran las figuras 2.3. y 2.4., que lanzó su sistema de asistencia a la conducción en el año 2015, cuya versión de Software propio era la versión 7. El nivel de asistencia que proporciona este sistema es total, pero requiere de la máxima atención del conductor, para tomar el control del vehículo en caso de ser necesario. De hecho, desde el mes de octubre del año 2016, todos los vehículos fabricados por esta marca llevan incorporado dicho sistema, y equipados con 8 cámaras, 12 sensores de ultrasonidos y un radar situado en la parte delantera que activa la conducción autónoma completa del vehículo. No obstante, dichos sistemas aún no tienen capacidad legal en España, por lo que estos vehículos lo llevan desactivado de manera temporal.

Esto es debido a que, en la actualidad, es necesario ajustar determinados aspectos relacionados con la seguridad vial, seguros de los automóviles y otros temas desarrollados en el apartado legal de este documento.

Fig. 2.3. Vehículo Autónomo de Tesla. *(muycomputer, 2018)*

Fig. 2.4. Vehículo Autónomo de Tesla: Interior. *(TicBeat, 2017)*

Por otro lado, las competiciones de vehículos autónomos son otra forma de motivar a los fabricantes a seguir desarrollando estos sistemas. La competición más conocida actualmente es el *DARPA Grand Challenge (American Defense Advanced Research Projects Agency)*, que cuenta con varias ediciones realizadas, y que fue fundada y organizada por el Departamento de Defensa estadounidense [10].

De esta competición salieron muchos de los investigadores actualesde las principales empresas que estudian este sector, como *Tesla* o *Google*, como explica “eldiario” en su web. Nadie consiguió terminar los 225 km de los que constaba la primera edición y en la segunda, un año después, lograron finalizar 5 vehículos, sin la interacción entre sí durante el recorrido.

Fig. 2.5. Stanley, campeón de la edición de 2005. *(Eduardo, 2017)*

2.2.3. Sistemas de ayuda a la conducción (*ADAS*)

Como ya se anticipaba de manera introductoria, a pesar de la seguridad que aportan las carreteras del mundo actualmente, los accidentes de automóvil son causantes de más de 3.500 muertes al día en todo el mundo.

Los sistemas de ayuda a la conducción o *ADAS (Advanced Drive Assistance Systems*), son sistemas capaces de detectar un entorno en continuo cambio a través del procesado de datos obtenidos por sensores y elementos de detección en tiempo real.

En la actualidad, los sistemas más conocidos son:

- Sistema de detección de peatones y obstáculos: son sistemas capaces de detectar obstáculos que se encuentren a una distancia determinada en la trayectoria del vehículo.
- Sistema de detección de ángulo muerto: muchos de los accidentes que se dan en las carreteras son debido a cambios de carril erróneos, es decir, cambios de carril en los que hay un vehículo que no ha sido visto por el conductor a través del espejo retrovisor. Estos sistemas alertan al conductor, por medio de una señal luminosa cuándo puede realizar ese cambio de carril de manera segura.


Fig. 2.6. Detector de ángulo muerto 1. *(20 minutos web, 2017)*

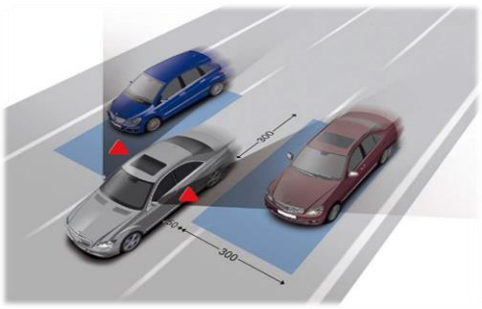
Fig. 2.7. Detector de ángulo muerto 2. *(Circula Seguro, 2017)*

- Control inteligente de los faros: a través del uso de sensores que permitan obtener la claridad del entorno, el vehículo es capaz de decidir el momento en que deben ser encendidas las luces de cruce.

- Asistente de detección de señales de tráfico: un sistema capaz de filtrar las señales en una base de datos según tamaño, color y forma, que permiten aclarar la finalidad de dicha señal.

Fig. 2.8. Detector de señales. *(Coches.com, 2018)*

- Sensores y sistemas de aparcamiento: los sensores de aparcamiento son uno de los sistemas más implementados, los cuales, son capaces de advertir al conductor de la proximidad de un obstáculo durante el aparcamiento.
- Sistemas de frenado automático: existen muchos vehículos en la actualidad que llevan incorporado un sistema capaz de detectar un obstáculo en la trayectoria del vehículo y de activar el freno de manera automática cuando la distancia a dicho obstáculo se reduce de manera considerable.

Estos son algunos de los sistemas utilizados, aunque existen muchos otros ya implementados o en desarrollo. Todos estos sistemas que se están creando, son aquellos que pueden crear en su conjunto la realidad de un vehículo autónomo.

Uno de los factores clave en la implementación de los sistemas de percepción consiste en el procesado de la información captada por los sensores. Por tanto, la disciplina fundamental para el desarrollo de cualquier sistema se centra en el análisis a través de la visión por computador.

2.2.4. Principales bases de datos

Uno de los principales problemas que se mantienen en la actualidad con la visión por computador en estos vehículos consiste en el reconocimiento y diferenciación de los objetos, tal y como explica Joel Janai. Al cabo de unos años de investigación, se ha llegado a la conclusión de que las tomas de referencia y bases de datos (*datasets)* almacenados a través de simulaciones realizadas con vehículos en todo tipo de entornos diferentes, como entornos con lluvia o niebla, con temperaturas altas y bajas, o con mayor o menor grado de iluminación, son una pieza clave para el entrenamiento del vehículo inteligente. A pesar de eso, el reconocimiento de superficies, objetos y contornos de interés de un entorno sigue siendo una tarea que requiere de métodos no automatizados bastante laboriosos.

Las principales *datasets* o bases de datos utilizadas son:

- Reconstrucciones tridimiensionales: es una ficha introducida por Scharstein & Szeliski en el año 2002 para obtener comparaciones del funcionamiento de diferentes algoritmos. El suelo verdadero se obtiene a través

de la reconstrucción de componentes planos en escenarios planos. Esta ficha utiliza el valor del error cuadrático medio y el porcentaje de pixeles erróneos entre la estimación y los mapas de disparidad obtenidos de manera real.

- *Optical flow*: esta base de datos introducida por Baker en 2011 utiliza el rastreo de texturas fluorescentes rociadas previamente sobre objetos del entorno. Con esto, se generan ocho secuencias formadas a su vez por ocho estructuras diferentes. Utilizando dos secuencias se pueden obtener los datos de interés.

- Reconocimiento y segmentación de objetos: lanzada por Everingham en 2010, es una base de datos que aplica un filtro de análisis inicial, seguido de una segmentación y por consiguiente el reconocimiento de los objetos del entorno.

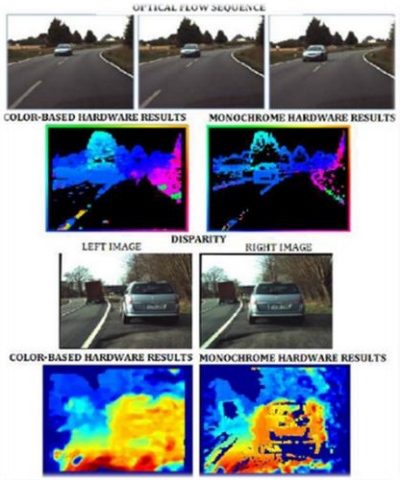

Fig.2.9. Mapeo con *optical flow*. *(Joel Janai, 2017)*

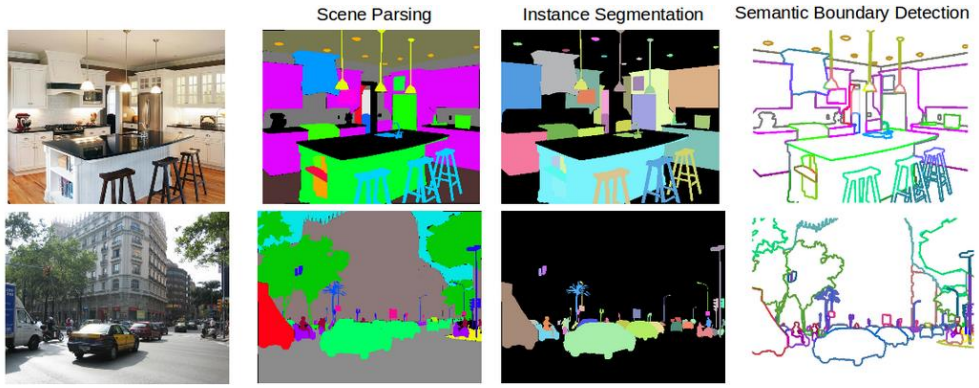

Fig. 2.10. Ejemplo de segmentación y reconocimiento de objetos. *(Joel Janai, 2017)*

- *KITTI (Karlsruhe Institute of Technology & Toyota Technological Institute) Vision Benchmark*: base introducida por Geiger en 2012 para el flujo óptico, visión y detección de objetos. Esta referencia utiliza pares de imágenes a una resolución determinada que obtiene la identificación de los objetos a través de la inserción de nubes de puntos en la imagen del entorno tridimensional [11].

Fig. 2.11. Ejemplo de mapeo con *KITTI Vision Benchmark*. *(Joel Janai, 2017)*

2.2.5. Tipos de componentes del vehículo autónomo

- Componentes Software:

Para una conducción automática en un entorno abierto, urbano e impredecible, es necesario contar con una serie de sistemas capaces de proveer de información al vehículo:

- Sistemas de localización:

Los sistemas más conocidos son el GPS (*Global Positioning System* o Sistema de Posicionamiento Global), que es actualmente el sistema más conocido y utilizado, o el RTLS *(Real Time Locating System*). Estos últimos, son sistemas de posicionamiento en tiempo real, basados en comunicación por radiofrecuencia [12].

- Sistemas de percepción del entorno y planificación:

Se refiere a los sistemas utilizados para procesar la información y proceder a la toma de decisiones en base a los resultados obtenidos.

- Sistemas de control global del entorno Software:

Los vehículos autónomos requieren de un sistema central que gobierne el vehículo, el cual tome el control en la toma de decisiones.

- Sensores y actuadores para la recogida de información relevante

Todo el sistema está basado en una creación de algoritmos de aprendizaje, los cuales, permiten al vehículo "aprender" a interpretar los datos recogidos por los sensores. También se utilizan modelos probabilísticos a través de métodos de predicción para conocer la evolución de un obstáculo en movimiento.

- Hardware de interés particular: **sensores**

Con el objeto de tomar decisiones de manera efectiva y acertada, el coche necesita recopilar toda la información que va captando a través de los sensores incorporados. Es necesario contar con un sistema de procesado de datos y ordenadores que ejecuten y recopilen la información.

El láser escanea el entorno haciendo sucesivos mapeos a través de los haces de luz calculando de manera continua la distancia de los objetos captados. Combinar esa información con la captada por las cámaras permiten construir entornos tridimensionales fiables.

2.2.6. Sensores

En este apartado, se comentan los diferentes sensores y dispositivos de medición y captación de información del entorno estudiado. Uno de los procesos de mayor importancia para la conducción autónoma efectiva consiste en la capacidad de

elaboración de un mapa del entorno y de localización de la posición del propio vehículo en dicho entorno. Para ello, son necesarios los sensores.

- **Cámaras:**

Por un lado, cabe destacar la presencia de las cámaras o sensores de imagen, en los vehículos del mercado actual. Cada vez, es más habitual encontrar automóviles cuyo equipamiento incorporan cámaras de visión trasera que se activan en la pantalla del navegador al momento de activar la marcha atrás. Esta implementación se realiza con el único fin de aportar información visual al conductor a la hora de estacionar el vehículo. El sistema consiste en captar y reproducir las imágenes de la cámara en la pantalla del vehículo.

También se instalan en la parte delantera de los vehículos con el mismo fin, o en el salpicadero para medir el comportamiento del conductor.

El procesado de los datos captados por los sensores y cámaras del vehículo es complejo. Esto se debe a la necesidad de creación de un sistema capaz de proyectar en dos dimensiones, un mapa que se corresponda con la realidad obtenida de un mundo tridimensional. Además, en caso de utilizar varios sensores, caso imprescindible en los vehículos autónomos, ese sistema debe ser capaz de distribuir la información de cada sensor en un sistema de coordenadas común.

Para esto, es importante la calibración, como se explica en el artículo de Janai. La calibración habitual de las cámaras implementadas en el vehículo se realiza a través del enfoque de un “tablero de damas”. Además de la calibración de las cámaras, hay otros aspectos como la velocidad y robustez para la adaptación al cambio de condiciones en las imágenes.

Una de las premisas del vehículo autónomo consiste en la maximización de la información del entorno, por lo que también se suelen utilizar cámaras omnidireccionales. Estas cámaras son capaces de captar información del entorno de manera panorámica o incluso en los 360 grados. En este caso, las herramientas de calibración son algo más complejas, porque no pueden consistir en el uso de elementos bidimensionales.

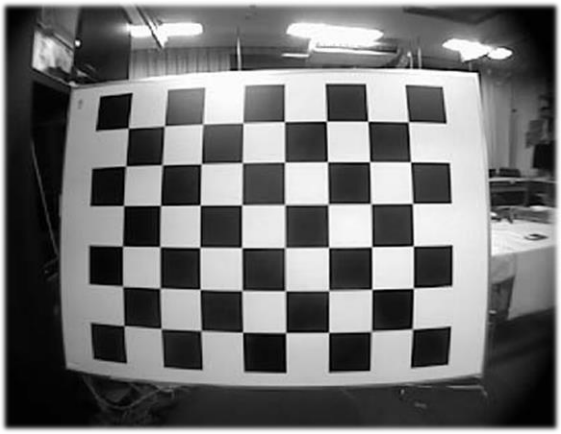

Fig. 2.12. Calibración de cámara Simple. *(Joel Janai, 2017)*

Fig. 2.13. Calibración de cámara omnidireccional. *(Joel Janai, 2017)*

- **Sensores de ultrasonidos:**

Estos sensores funcionan a través de la transmisión de ondas de ultrasonido. El sensor emite señales de manera continua y es capaz de medir la distancia a un objeto detectado en función del tiempo que tarda en recibir la onda enviada. El ejemplo más común, como se aprecia en la figura 2.14, son los sensores que forman parte del sistema de aparcamiento, debido a que su margen de error se minimiza conforme disminuye la velocidad y la distancia al objeto.

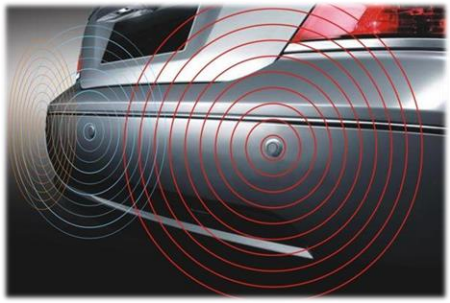

Fig. 2.14. Sensores de ultrasonidos. *(Automoción Total, s.f.)*

- **Radares:**

Son sensores de radio, que funcionan con ondas electromagnéticas y son muy utilizados en la tecnología aviónica y marítima (Fig. 2.15). Dichas ondas, se emiten con el fin de obtener la reflexión de dicha señal y con ello, percibir la información del objeto detectado en cuanto a distancia y velocidad de aproximación se refiere. El funcionamiento de estos sensores es similar al de los sensores de ultrasonidos, con la diferencia de que son más útiles en distancias lejanas y velocidades más altas [13].

Fig. 2.15. Radar utilizado en aviones y barcos. *(La Razón Digital, 2018)*

El principal hándicap de estos sensores es que trabajan con información bidimensional, aportando falta de información de los objetos detectados a medida que aumenta la altura del objeto.

Sin embargo, en la actualidad, se trabaja en la investigación y desarrollo de sensores electromagnéticos que obtengan la información tridimensional de un entorno, con el objetivo de obtener la mayor cantidad de información posible.

Estos sensores son muy conocidos en el ámbito automovilístico también, siendo una herramienta habitual para que los organismos de seguridad de los países mantengan un entorno seguro en la conducción. Esto es debido a su capacidad de detección de objetos y de la velocidad con la que se mueven.

- **Lídar[1]:**

El lídar es un sensor láser capaz de detectar el entorno en un ángulo completo de 360º a través de la emisión y reflexión de millones de haces de luz pulsada, enviados de forma continua. Este sensor define los objetos detectados en un entorno formando nubes de puntos. Cada vértice de esa nube de puntos se corresponde con un haz de luz pulsada enviada por el sensor lídar y que posteriormente ha recibido reflejada [14].

[1] Lídar o lidar: Del acrónimo *LIDAR* (*Light Detection and Raging* o *Laser Imaging Detection and Ranging*).

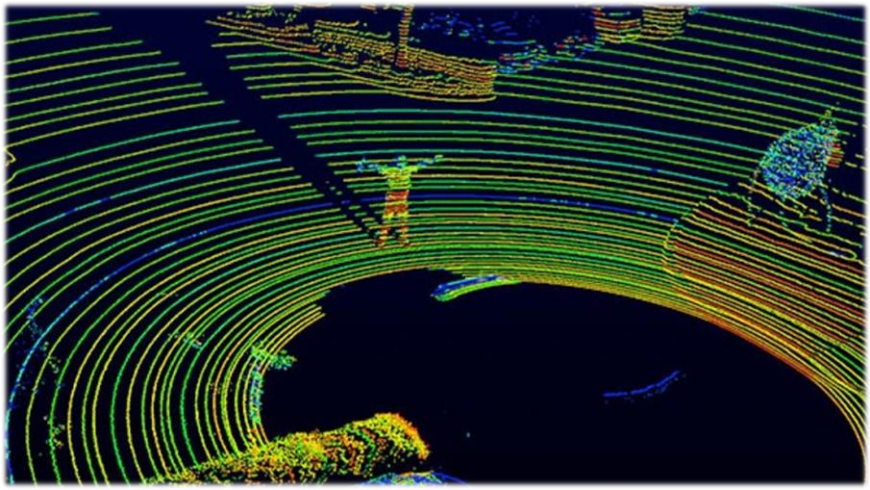

Fig. 2.16. Detección de un sensor lídar. *(Ramsey, 2017)*

Es interesante comentar en la imagen anterior, la detección de la persona que aparece, cuya sombra se aprecia como una sombra, es decir, una zona sin información por parte del sensor lídar debido al obstáculo vertical que representa el peatón.

2.2.7. El vehículo autónomo *IVVI 2.0*

Actualmente, en la universidad, existen tres proyectos en ejecución diferentes del vehículo autónomo, y un proyecto completado:

- *iCab* (*intelligent Campus* Automobile): carros de golf de conducción autónoma definida para el traslado de visitantes por las diferentes zonas de la universidad. Con esta plataforma, se investiga actualmente la localización, el mapeo del entorno, sistemas de planificación de ruta, detección y clasificación de objetos, y la coordinación entre sistemas.

- *SkyOnix*: dron en el cual se están implementando sistemas de detección y clasificación de objetos, y sistemas de navegación aplicables en la conducción autónoma.

- *IVVI 2.0*: vehículo autónomo evolución del primer prototipo: el *IVVI 1.0*, proyecto completado que se mencionaba anteriomente.

Fig. 2.17. Vehículos Autónomos de la Universidad Carlos III de Madrid. *(Intelligent Systems Laboratory, 2016)*

El proyecto desarrollado en esta memoria está pensado particularmente para su posterior implementación en el *IVVI 2.0*, mostrado en la esquina inferior derecha de la figura 2.17.

El *IVVI 2.0* es un vehículo autónomo, implementado con el principal objeto de investigar los *ADAS* (*Advanced Driver Assistance Systems*), sistemas avanzados de asistencia a la conducción.

La universidad, explica en su web [15], que el vehículo está diseñado como vehículo autónomo para el mercado del futuro, con los sensores y sistemas de actuación integrados de manera invisible. Los elementos que van implementados son:

- Sistemas de visión para carreteras, detección y clasificación de objetos y señales de tráfico probados en condiciones de conducción diurna.
- Una cámara implementada delante del espejo retrovisor interior para la detección de peatones en condiciones de conducción nocturna.
- Un sensor láser multicapa instalado en el paragolpes delantero del vehículo para detección de objetos.
- Un dispositivo de detección de movimiento en el salpicadero para el control facial del conductor.
- Un sistema implementado por comunicación *CAN*[2] para el análisis de los comportamientos del conductor, con el uso de un dispositivo *Kinect*, el cual consiste en una cámara *RGB* (*Red – Green – Blue*), un sensor de profundidad, micrófono y un procesador personalizado [16]. Dicho dispositivo, es compatible con el SO *Linux*, por lo que su integración es sencilla.
- Un receptor *GNSS* (*Global Navigation Satellite System*), es decir, un sistema de navegación satélite utilizado para el posicionamiento y localización del vehículo, y una *IMU* (*Inertial Measurement Unit*) situados en el techo del mismo con el fin de obtener información sobre el posicionamiento.

Además, incorpora varias cámaras en las zonas laterales delanteras del vehículo, que aportan información sobre la detección de peatones y obstáculos, en un vehículo diseñado con soporte de la arquitectura ROS. Este vehículo es capaz de llevar a cabo una conducción autónoma con detecciones en tiempo real.

2.2.8. Sistemas de visualización

Los sistemas de visualización en la conducción autónoma de vehículos son aquellos que pueden aportar confianza al conductor acerca de la situación del coche durante el recorrido. El conductor, al no ser el responsable del manejo del vehículo puede tener, en ciertos momentos, la preocupación de cómo actuará el vehículo.

Por ello, estos sistemas también tienen importancia en el vehículo autónomo. A menudo, la reacción del conductor a los avisos o alarmas que el automóvil puede activar para informar de alguna circunstancia, es desconocida.

Por ejemplo, en un sistema de detección de carril, si el conductor rebasa la línea que limita un carril de manera involuntaria, dicho sistema puede advertir a través de una

[2] Del acrónimo *Controller Area Network*, es un protocolo de comunicaciones diseñado por la marca *Robert Bosch* para la transmisión de mensajes en entornos distribuidos.

vibración en el asiento, o de una alarma acústica. En estos casos, la reacción del conductor es imprevista y la corrección de dicho problema puede conllevar situaciones más comprometidas.

Por tanto, los sistemas de visualización pueden ser también una solución a este problema. Dichos sistemas están formados, únicamente, de una pantalla que muestra la información relevante de la conducción del vehículo, lo cual, como se explica en párrafos anteriores, puede aportar confianza y seguridad al ocupante principal del vehículo.

La empresa *Nvidia* es uno de los principales motores actuales de estas tecnologías, las cuales están implementadas en vehículos de las mejores marcas del mundo: *Porsche, Audi, BMW, Tesla*…etc.

Fig. 2.18. Tecnología de visualización de Tesla basada en Nvidia. *(Nvidia, 2018)*

Fig. 2.19. Tecnología Porsche basada en Nvidia. *(Nvidia, 2018)*

Por tanto, estas tecnologías, podrían aportar la seguridad suficiente a los ocupantes de un vehículo manejado de manera autónoma.

2.2.9. *Android Auto*

El sistema operativo Android ha sido el sistema más conocido y utilizado en las últimas décadas con el nacimiento de los *smartphones*, los télefonos de última tecnología que ya funcionan como pequeños ordenadores. Debido al avance en el vehículo autónomo, la llegada de Android a este sector ya se ha llevado a cabo.

El uso de teléfonos móviles y *tablets* es algo cotidiano y que utiliza la gran mayoría de la población mundial. El trabajo con los móviles cada día es más común, y debido a su tamaño y capacidad de procesado, es una herramienta útil a la hora de visualizar los algoritmos captados por los sensores y cámaras de un vehículo inteligente en la fase de desarrollo de los sistemas de percepción.

La llegada del sistema operativo más utilizado en los últimos años al sector automovilístico ya se ha producido, como se ha comentado anteriormente de manera introductoria.

Android Auto es, según la web oficial de la plataforma, un sistema creado para el sector del automóvil pensando en la seguridad de sus ocupantes [17]. Explican, que está formado por una interfaz gráfica sencilla e intuitiva con controles integrados en el volante del vehículo. Además, cuenta con alarmas potentes que advierten al conductor evitando las distracciones del mismo en la carretera.

Son muchas las marcas que ya llevan integrado este sistema, para el que es necesario contar con un teléfono móvil con sistema operativo compatible cuya versión mínima es la 5.0 (*Lollipop).*

Este producto lanzado por *Google* consiste en una aplicación que permite al vehículo acceder a otras *apps* pertenecientes al grupo *Google*, como *Google Maps*, aplicaciones de música, y además permite acceder por comandos de voz a las opciones de llamada [18].

Fig. 2.20. Sistema Android Auto. *(Android, 2018)*

Desde el nacimiento de este sistema en el año 2014 hasta la fecha, ya son muchas marcas las que colaboran con este estándar de *Google* integrado en sus automóviles.

2.2.10. Plataforma *Apple* para la conducción autónoma

Apple, la empresa líder en telefonía actualmente, también mostró interés en la creación de un vehículo autónomo. Ese proyecto recibió el nombre de "*Project Titan*", pero, hace año y medio, la cadena internacional *Bloomberg* mostraba el cambio de idea de la empresa de Cupertino [19].

En octubre de 2014, se dio a conocer el cambio de dirección en cuanto a la investigación del vehículo autónomo por parte de la empresa estadounidense, que expresaba la decisión de centrarse en el desarrollo de un Software competitivo y compatible con las tecnologías actuales implementadas en este sector del vehículo autónomo.

Sin embargo, a principios del año 2018, se ha dado a conocer de manera oficial el interés de la empresa por retomar la idea del "*iCar*", aclarando que actualmente cuentan con una gran confianza obtenida en base a la inteligencia artificial y el método "*machine learning*", es decir, entrenamiento y aprendizaje del vehículo [20]. Su tecnología, estaría basada en la captación del entorno a través del uso de radares lídar tridimensionales.

3. RECURSOS UTILIZADOS

En este apartado se mencionan y explican los recursos utilizados para el desarrollo del proyecto y se definen conceptos que permitirán la comprensión del mismo. Además, se exponen los motivos por los cuales se ha decidido utilizar dichos recursos.

3.1. Sistema operativo *Ubuntu Linux*

Este sistema operativo es un sistema de código abierto utilizado en computación. Es una plataforma muy utilizada en el ámbito de desarrollo Software en gran parte por la facilidad de uso a través del terminal. En la actualidad, es un sistema que carece de virus, lo que aporta al programador una confianza en la seguridad de su ordenador y su trabajo.

Uno de los principales motivos que ha llevado a decidir el uso de este sistema operativo ha sido el uso de este SO del Software libre, algo bastante importante a la hora del desarrollo de aplicaciones y sistemas.

Además de esto, otro de los motivos importantes ha consistido en la posibilidad de utilización de aplicaciones de *Qt*, entorno que se explicará en el subapartado 3.3. de éste mismo capítulo. Esta opción es posible a partir de la versión 11.04 proporcionada por el suministrador oficial, facilitando así la integración de aplicaciones con extensión de *Qt.*

Por último, y la motivación más importante para la decisión de usar este sistema operativo, es la posibilidad de trabajar con ROS, entorno que se explica en el siguiente apartado y sistema base del proyecto que se expone en el presente documento.

3.1.1. Introducción a *Ubuntu*

Este sistema operativo nació del código base del proyecto *Debian*[3], a través del cual se buscó mejorar el actual y corregir errores, haciendo así que la navegación y manejo del sistema fuese más asequible [21].

Los trabajadores implicados en el desarrollo de este sistema comprendieron la complejidad y fragmentación que sufría la gestión de paquetes y aplicaciones, por lo que crearon el Centro de Software de Ubuntu, un sistema capaz de proporcionar aplicaciones y repositorios de manera fácil y rápida al usuario. Este cambio en el año 2009 fue el primero de muchos que han llevado al sistema operativo a ser el principal sistema de desarrollo para programadores, como explica el artículo publicado en la web *Wikipedia.*

3.1.2. Instalación del SO

La instalación del sistema operativo proporcionado por la comunidad de *Ubuntu* conlleva la mayoría de las veces la creación de una partición del disco duro, en caso de querer mantener el SO inicial que suele estar instalado en el ordenador en el momento de su venta.

[3] *Debian* es una comunidad formada por desarrolladores y usuarios que mantiene un sistema operativo de Software libre que da soporte a múltiples arquitecturas de computación.

Los paquetes necesarios se pueden descargar en la web oficial de la comunidad. En caso de contar con dos o más sistemas operativos en el mismo ordenador, el ordenador mostrará una pantalla inicial antes de arrancar uno de los SO instalados, mostrando las diferentes opciones para facilitar la selección.

3.2. ROS (*Robot Operating System*)

ROS es un *framework*[4] usado en el desarrollo Software para robots, el cual provee la funcionalidad de un sistema operativo en un clúster[5] heterogéneo [22]. Este sistema se definió en el año 2007, llamado *Sitchyard* por el Laboratorio de Inteligencia Artificial de Stanford. Fue creado para dar apoyo a un proyecto robótico que desarrollaban en ese año en la Universidad.

En estos últimos años la arquitectura de ROS se ha establecido como el sistema estandarizado para el desarrollo de sistemas robóticos controlados, debido a las principales ventajas que otorga con su dinámica de desarrollo bajo Software libre.

Además, los diseños realizados se utilizarán en el *IVVI 2.0*, cuyo funcionamiento es controlado con ROS. Debido a esto, y en busca de una utilidad funcional de las implementaciones realizadas en este proyecto, es imprescindible el uso de esta arquitectura.

3.2.1. Introducción a ROS

Se puede definir como un sistema que provee de librerías y herramientas al desarrollo Software en proyectos en los que intervienen elementos robóticos [23]. La web oficial de este sistema explica que provee al usuario de los servicios estándar de un sistema operativo, tales como el control de dispositivos de bajo nivel y la implementación de funcionalidad de uso común, como la transmisión de mensajes entre sistemas.

Este sistema está basado en la Teoría de Grafos[6]. En él, el procesamiento de la información se ejecuta en nodos capaces de recibir, transmitir y multiplexar mensajes de todos los sistemas electrónicos utilizados en un proyecto, tales como robots, sensores, actuadores, cámaras…etc.

3.2.2. Instalación del entorno

La instalación de este sistema se realiza en un sistema operativo *Linux*, dado que sus librerías están orientadas para ese tipo de sistemas. A través de comandos guiados en la web oficial de ROS, es posible instalar el entorno básico para comenzar a trabajar.

---

[4] *Framework* es una estructura conceptual y tecnológica con un soporte definido, normalmente con artefactos o módulos de Software concretos, que puede servir como base de datos para la organización y el desarrollo Software.

[5] Del ingés *cluster*, se aplica a los conjuntos o conglomerados de ordenadores unidos entre sí normalmente por una red de alta velocidad y que se comportan como una unidad indivisible.

[6] La Teoría de Grafos es una ciencia basada en las matemáticas e informática, que estudia la propiedad de los grafos. Estos, están definidos como vértices o conjunto de nodos unidos por enlaces llamados aristas, que permiten representar relaciones binarias entre diferentes elementos de un conjunto.

No obstante, es necesario comentar que algunos de los comandos necesarios para la instalación de diversos paquetes están obsoletos para la última versión disponible de *Ubuntu Linux*, sistema operativo utilizado para el proyecto.

Para comprender el funcionamiento de ROS, ha sido necesario el trabajo durante algún tiempo con la herramienta realizando pequeños ejemplos proporcionados por el desarrollador en su web y otros proporcionados por el departamento de la Universidad.

3.2.3. Descripción de elementos

Por tanto, para comprenderlo mejor, es necesario conocer los conceptos más importantes en los que se basa esta estructura:

- *Workspace:* para el correcto funcionamiento de la herramienta, es necesario crear un espacio de trabajo en el que almacenar toda la información utilizada. La creación de este espacio genera de manera automática algunos directorios necesarios dentro del mismo.

- *Packages*: los paquetes de ROS son la unidad principal de organización. Dicho paquete debe contener procesos ejecutables en tiempo real, una librería con ficheros que definan al paquete y otros elementos que sean necesarios para su correcto funcionamiento.

- *Nodes*: son procesos de diseño computacional. Para un sistema de control básico, son necesarios varios nodos. Por ejemplo, un nodo puede controlar un sensor láser, otro los motores que lo mantienen activo…etc.

- *Topics*: los *topics* son los contenedores de información relevante que, en determinados momentos, es necesario comunicar entre nodos.

El paso de los *topics* entre nodos, es decir, el paso de información, se realiza a través de *Subscribers* y *Publishers.* Son respectivamente, los elementos que obtienen la información de un nodo y los que posteriormente la publican.

- *Services*: existen varios métodos de obtención y paso de la información. Pero, en caso de ser necesaria la transmisión de varios mensajes al mismo tiempo, se utilizan los servicios.

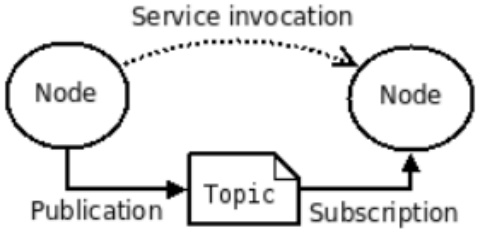

Fig. 3.1. Estructura del funcionamiento básico de ROS. *(ROS, s.f.)*

- *Bags:* son ficheros llamados así por su extensión "*bag*" y son capaces de almacenar los mensajes captados por los diferentes sensores de un sistema robótico.

- *Messages:* Los nodos de ROS se comunican entre sí para la transferencia de información en un entorno robótico. Son una estructura de datos que comprende campos escritos, mensajes con variables de tipo entero, booleanos, flotantes …etc., y también matrices de estos tipos. También pueden comprender estructuras anidadas como los *arrays,* comúnmente conocidos en lenguaje C.

- Repositorios: son archivos o ficheros donde se puede almacenar información de manera abierta y conjunta, cuya funcionalidad es similar a la de las bases de datos. Son, en la arquitectura de ROS, fuentes de información muy importantes para los desarrolladores de sistemas robóticos.

Estos son los principales elementos a conocer para comprender el funcionamiento de ROS, pero el más importante es el siguiente:

- *Master*: es el encargado de proporcionar un nombre de registro y una búsqueda para el resto del conjunto computacional. Sin él, los nodos no tienen conexión entre sí y no son capaces de obtener y transmitir información.

Los conceptos descritos previamente han sido los utilizados en el desarrollo de este proyecto. No obstante, existen otros conceptos muy utilizados como:

- *Metapackages*
- *Packages Manifests*
- *Clients*
- *Etc.*

3.3. *Qt Creator*

Este entorno de desarrollo integrado (IDE) es un entorno multiplataforma creado en base a código C++, *Javascript*[7] y *QML*[8]. Es parte de un kit de desarrollo de Software principalmente utilizado en el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI) a través de las bibliotecas *Qt.*

Además, esta plataforma es soportada por los principales sistemas operativos del mercado: *Windows* de Microsoft, *Mac Operative System* del fabricante Apple y *Linux.* Este último es el utilizado en la ejecución del proyecto, por lo que se convierte en un motivo de interés para ser utilizado.

Por otro lado, uno de los lenguajes soportados por el entorno es C++, lenguaje estudiado durante el grado, por lo que es otra motivación importante por el cual se ha usado.

[7] *Java Script* es un lenguaje de programación interpretado orientado a objetos, similar a C++.
[8] *QML, Qt Meta Languages* es un lenguaje basado en *Java Script*.

3.3.1. Instalación de la plataforma

La instalación del entorno de desarrollo mencionado se realiza a través del terminal de *Ubuntu* mediante la introducción de un único comando o a través del Centro de Software proporcionado por el sistema operativo.

3.4. *Android Studio*

El entorno de desarrollo conocido como *Android Studio* es un programa perteneciente al sistema operativo *Android*, y que permite el desarrollo de aplicaciones para móviles, *tablets* y *smartwatches* (pulseras de actividad) a través de la programación en lenguaje *Java*.

La creación de una aplicación que permita visualizar el entorno captado por los diferentes sensores y cámaras implementadas en el vehículo es un proyecto viable para ser instalado en los vehículos con sistemas de detección o vehículos autónomos. Lo cual, ligado al hecho comentado en el capítulo 2.2.9., en el que se expone la existencia del sistema *Android Auto* lanzado por *Google* compatible con el entorno comentado, se ha convertido en el motivo principal por el cual se crea la aplicación en este sistema operativo.

Además, la posesión particular de un dispositivo móvil, *smartwatch* y *tablet* con el sistema operativo en cuestión, han facilitado el trabajo e implementación del proyecto. Por tanto, esto también se convierte en un motivo de peso a tener en cuenta.

3.4.1. Introducción

Este entorno se convirtió así, en el IDE (*Integrated Development Environment*) oficial de la plataforma, sustituyendo a *Eclipse*, anterior entorno utilizado por *Android* para el desarrollo de aplicaciones, según un artículo publicado en la web *Wikipedia* [24].

A pesar de ser lanzado de manera oficial a finales del año 2014, desde el año anterior ya se conocía su existencia, y a mediados de 2014 fue lanzado como plataforma en fase de pruebas.

3.4.2. Instalación del entorno

Este entorno se puede instalar tanto para el sistema operativo *Linux* como *Windows*, pero debido al uso compartido con ROS, fue necesaria su instalación en el primero. A través de unos pasos proporcionados por la web oficial de la plataforma, la instalación es rápida y sencilla.

Los proyectos en esta plataforma están estructurados de manera que, por un lado, se encuentra el diseño gráfico de la propia aplicación, y por otro, la programación que permite actuar de diferentes modos a la *app* en función de su uso.

3.4.3. Descripción de elementos

Android Studio posee una interfaz gráfica que permite comprobar el diseño de aplicaciones de dos modos, los cuales aparecen englobados en un directorio llamado "*Layouts"* o "Diseños":

- El diseño se puede realizar de manera gráfica sobre un emulador previamente seleccionado entre una amplia gama de teléfonos y otros dispositivos con sistema operativo compatible. La forma de actuar consiste en buscar los elementos que se quieren colocar en el diseño de la interfaz y arrastrarlos a un lugar determinado, atribuyéndole determinadas características en un desplegable mostrado en la parte lateral derecha de la aplicación como tamaño, color, nomenclatura…etc.

- También se puede llevar a cabo la integración de la aplicación a través de la implementación de código equivalente a la opción comentada previamente.

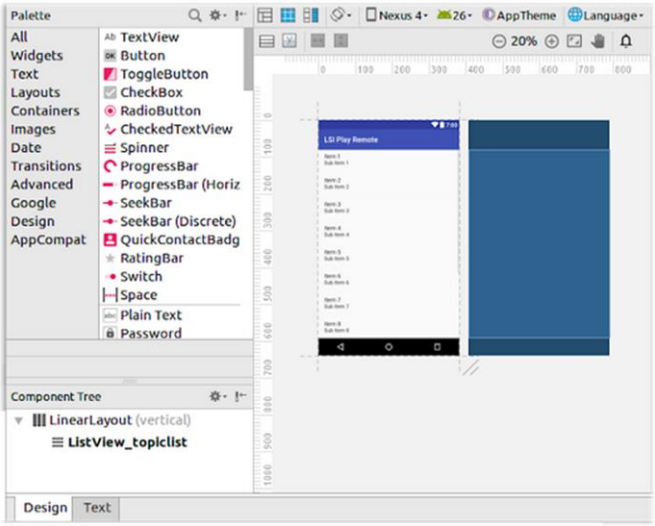

Fig. 3.2. Apartado de diseño de Android Studio

Fig.3.3. Diseño de la aplicación a través de código

Por otro lado, existe otro directorio en el que se encuentran los ficheros de programación *Java* llamado *"src*[9]*"*. En estos ficheros se da la funcionalidad a la aplicación y se pueden definir dos tipos principales de elementos:

- Por un lado, se pueden definir clases, cuya implementación puede tener soporte de librerías externas que se importan en el propio fichero de definición.

- La otra opción, es la de definir e implementar actividades o *activities*. Las actividades son clases que contienen su propia interfaz, de manera que el usuario puede interactuar con ella durante la ejecución de la aplicación en desarrollo.

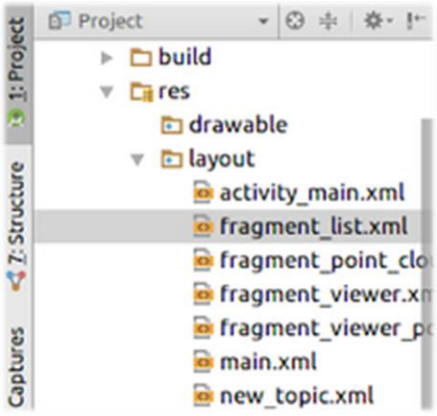

Fig. 3.4. Jerarquía de *Layouts*

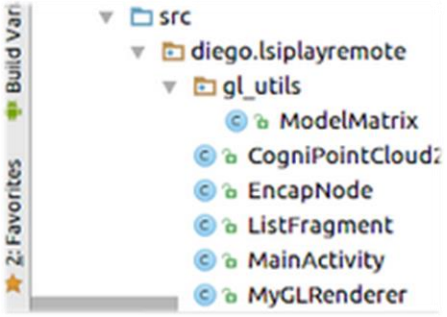

Fig. 3.5. Jerarquía *src*

El entorno de desarrollo de *Android* posee además un depurador con el cual es posible configurar un teléfono, *tablet* o *smartwatch* virtual. Con ese depurador se puede comprobar la funcionalidad de la aplicación en su fase de implementación. Otra opción disponible, consiste en conectar un teléfono móvil con su sistema operativo para depurar la aplicación de manera más rápida, con la versión del SO mínima especificada en los archivos de configuración del proyecto que contiene la aplicación a desarrollar.

Fig. 3.6. Depurador

Para ello, es necesario activar el modo "Depuración de USB" (*Universal Serial Bus*) disponible en el menú oculto de desarrollador, el cual, se puede desbloquear fácilmente con el uso de determinados comandos disponibles en las instrucciones del dispositivo móvil.

3.5. *Git*

Git es un Software de control de versiones diseñado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

[9] *src* se refiere a la abreviación de *source*, cuyo significado en castellano es "método".

El diseño de esta herramienta resulta de la experiencia del diseñador de Linux, quien, manteniendo una enorme cantidad de código distribuido y gestionado por mucha gente, y que incide en numerosos detalles de rendimiento, creó *Git* para gestionar mejor esa gran cantidad de información [25].

En el caso concreto de este proyecto, *Git* ha sido usado complementado por la web proporcionada por *Bitbucket*[10], web que permite:

- Almacenar repositorios manteniendo la jerarquía que adopta el *Git* subido.
- Comprobar diferencias entre diferentes versiones de un mismo proyecto.
- Recuperar el repositorio de manera segura en caso de tener problemas con el PC (*Personal Computer*).

Estos son los tres principales motivos por los cuales se ha utilizado durante el desarrollo del código, ya que, proporciona seguridad y un sistema de control de versiones de gran utilidad para los desarrolladores durante la depuración de los sistemas en implementación.

A través de la aplicación de comandos vía terminal, tal y como explica el creador de este formato en su web oficial de manera resumida, es posible administrar y asegurar los archivos con la web de almacenamiento comentada [26].

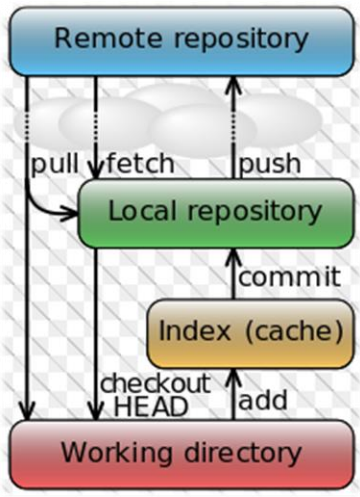

Fig. 3.7. Arquitectura de Git. *(Wiki Books, s.f.)*

3.6. Librerías de *OpenGL*

Las librerías *OpenGL (Open Graphics Library)* son las librerías que se han utilizado en el mapeo y procesado de imágenes y nubes de puntos para la aplicación, debido a que son una especificación estandarizada y soportada por múltiples plataformas utilizadas en la creación de aplicaciones que contienen gráficos bidimensionales y tridimensionales [27].

[10] *Bitbucket* es un servicio de alojamiento en red proporcionado por la empresa *Atlassian* para proyectos que siguen la estructura del sistema de control de revisiones *Git.*

Provee al desarrollador de más de 250 métodos para imprimir por pantalla dibujos y formas complejas totalmente personalizables en estilo, color, orientación, vista…etc., como se observa en las imágenes mostradas a continuación:

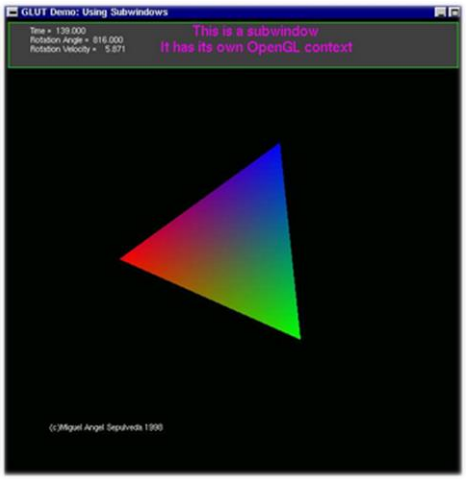

Fig. 3.8. Figuras con *OpenGL 1. (Sepúlveda, 2016)*

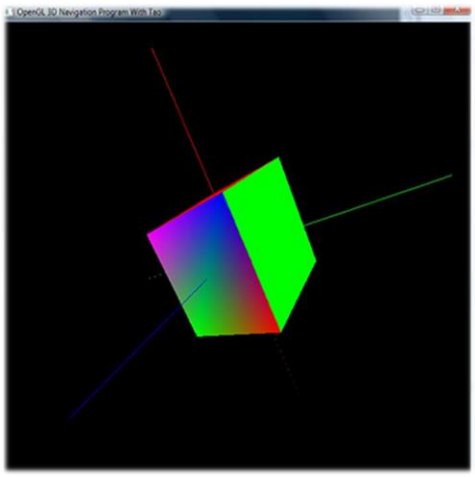

Fig. 3.9. Figuras con *OpenGL 2. (Quant Labs, 2014)*

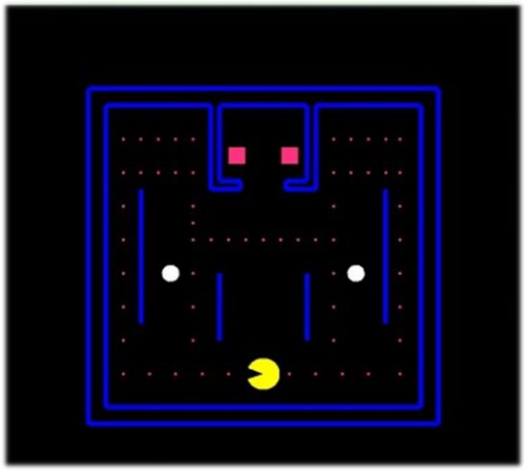

Fig. 3.10. Ejemplo de aplicación con *OpenGL. (Mygnet, 2015)*

4. DESCRIPCIÓN DEL PROYECTO

En este apartado se desarrolla de manera amplia y detallada en qué ha consistido el proyecto. También se explican las partes que lo componen, su funcionamiento y las bases que se han seguido para la consecución de los resultados. Por último, existe un apartado en el que se mostrarán simulaciones y resultados de todas las partes y los principales problemas encontrados durante el desarrollo de las mismas.

Existen dos partes diferenciadas que se han realizado para la obtención de los resultados finales:

- Visualizador de nubes de puntos
- Aplicación Android

4.1. Visualizador de nubes de puntos

Una nube de puntos está formada por vértices (o puntos) que dibujan, en un entorno estudiado, la superficie exterior de los elementos del entorno en un sistema de coordenadas tridimensional. Son el conjunto de información que un escáner láser capta de un entorno tridimensional de manera automática.

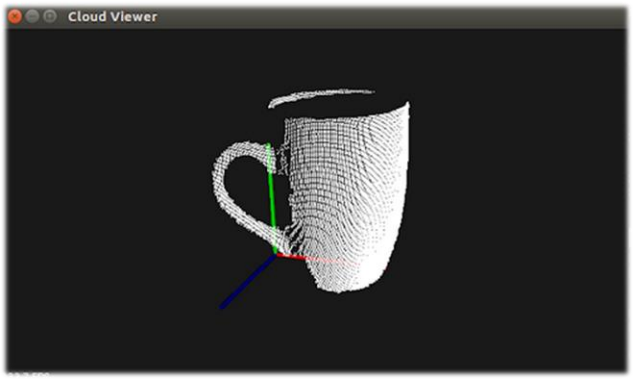

Fig. 4.1. Ejemplo de nube de puntos: Taza.

Las plataformas utilizadas para la creación y depuración del visualizador son:

- SO *Ubuntu Linux*
- ROS
- *Qt Creator*

El visualizador de nubes de puntos se crea para observar de manera gráfica la información captada por los sensores y almacenada en un fichero *bag*, compatible con la arquitectura de ROS. La función del visualizador es la de imprimir por pantalla el resultado obtenido del procesado de los archivos volcados al *bag* durante una simulación de detecciones del entorno del vehículo autónomo.

En caso de que ese archivo esté formado por información captada por un sensor en movimiento, la imagen resultante es diferente con el paso del tiempo, por lo que el

resultado debe ser una sucesión de imágenes que formen en su conjunto un vídeo del entorno recogido.

Ese efecto se consigue a través de la implementación de un método en el código utilizado que sea capaz de sustituir la imagen de *PCL* (*Point Cloud*) mostrada por una nueva cada vez que se compruebe la presencia de esa nueva nube de puntos.

Para la creación del visualizador, es necesario crear un espacio de trabajo nuevo con la jerarquía de directorios requerida por ROS. Utilizando los comandos vía terminal proporcionados por la web oficial del entorno robótico se obtiene un espacio de trabajo vacío y con los directorios internos necesarios para el funcionamiento del visualizador.

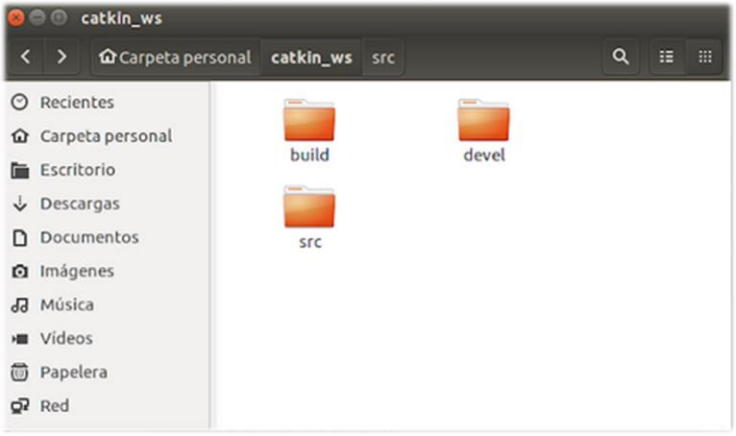

Fig. 4.2. Espacio de trabajo del visualizador.

Una vez realizado este paso, el siguiente consiste en la implementación del código para la visualización de las nubes de puntos.

Se parte de una base de código ya creado, proporcionado por la web oficial de *Point Clouds* [9], en el cual se pueden observar nubes de puntos que no varían en el espacio temporal. Siguiendo el código expuesto por la web, se obtiene el visualizador base sobre el que se realizan los cambios oportunos para el funcionamiento que se requiere.

El código está diferenciado por tres partes:

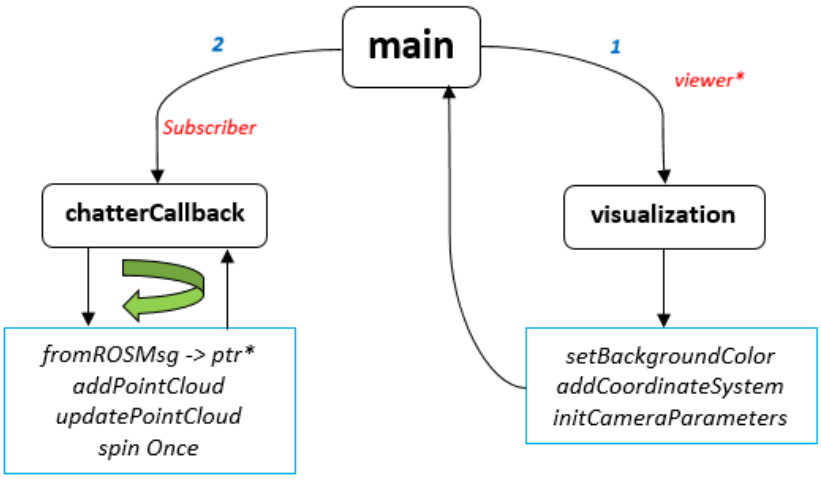

Fig. 4.3. Diagrama del flujo de datos del *3D Viewer*.

- **Método principal o *main*:**

En este método se crean los objetos necesarios para el funcionamiento del visualizador. En este método principal se crea un visualizador a través de un objeto puntero de ese tipo, asignándole un nombre determinado, el cual, aparecerá en la cabecera del visualizador cuando se ejecute. También se genera y ejecuta desde el principio del método un nodo, que será el contenedor para transportar la información deseada.

En segundo lugar, se asignan los valores requeridos a las variables de color del visualizador. Para ello, se hace una llamada al método de asignación de color predeterminado a partir del puntero previamente creado. Por otro lado, se hace la misma llamada al método que asigna el eje de coordenadas al visualizador y a un método que asigna unos valores por defecto a los parámetros de la cámara por la que se mostrarán los datos obtenidos.

El siguiente paso consiste en utilizar una variable global de tipo *Subscriber* para asignarle un nodo de ROS capaz de obtener la información de tipo *PointCloud2* del *bag* en ejecución, asignando unos parámetros que permitan al nodo utilizar el método *chatterCallback.*

Por último, se utiliza el método *spin* proporcionado por la librería *roscpp*[11], que mantiene la ejecución del nodo activa de manera continua hasta que se quiera terminar con la visualización.

- **Método de creación de la interfaz:**

En este espacio, el cual se corresponde con el bloque "*Visualization*" del diagrama mostrado en la figura 4.3., se inicializa cada una de las variables necesarias para la creación del visualizador. Se asignan valores de inicialización a los ejes de coordenadas, los colores de fondo del visualizador y se ejecutan métodos de inicialización necesarios para el correcto funcionamiento. Esto se realiza a través de una variable global utilizada de tipo puntero para llamar a los métodos necesarios.

- **Método para rellenar el visualizador con la información captada:**

Para terminar, se utiliza un método llamado chatterCallback o de re-llamada, en el cual se asignan los parámetros necesarios para que el funcionamiento del visualizador sea continuo.

En primer lugar, es necesario realizar la conversión de los mensajes obtenidos de tipo nube de puntos en un tipo que sea conocido por el sistema utilizado. Para ello, se crea un puntero con estructura de coordenadas cartesianas, al que se pasa la información convertida previamente por un método proporcionado por las librerías de nubes de puntos (librerías *PCL*).

Y, en segundo lugar, se utiliza una sentencia condicional *if* para actualizar el dato obtenido siempre que el visualizador reciba una nube de puntos nueva. Por último, para

[11] *roscpp* es una librería de ROS que permite crear objetos para publicar y obtener información a través de la ejecución de nodos.

que la actualización sea posible, se le añade un método a esa sentencia que mantenga el bucle de actualización de nubes de puntos con una frecuencia de 100 milisegundos.

4.2. Aplicación Android: *LSI Play Remote*

Para esta segunda parte del proyecto, se han utilizado los siguientes recursos:

- *Ubuntu Linux*
- ROS
- *Android Studio*
- *Git*
- Librerías *OpenGL*

4.2.1. Introducción

La creación de esta aplicación con *Android* se realiza, además de con el fin de facilitar el trabajo para los responsables del desarrollo del vehículo inteligente, para dar un paso más en la visualización de la información captada.

4.2.2. Diagrama de bloques de la aplicación

Para facilitar la comprensión del funcionamiento de la aplicación y su desarrollo, se muestra a continuación un diagrama de bloques sobre el que se basa la explicación de esta implementación.

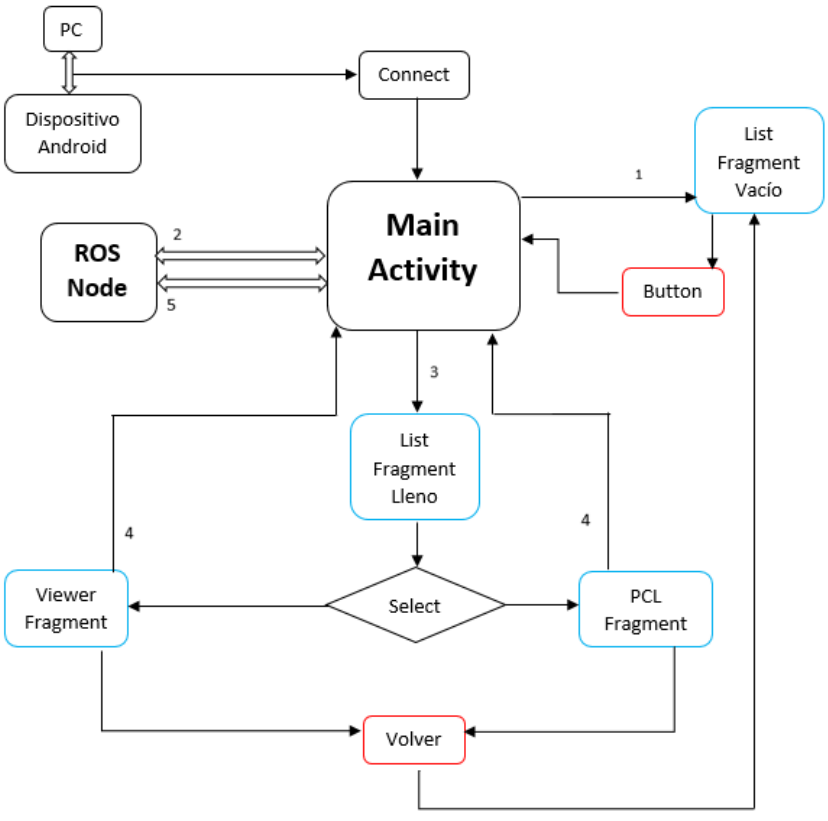

Fig. 4.4. Diagrama de funcionamiento de la *LSI Play Remote App*.

4.2.3. Desarrollo y funcionamiento de la aplicación

Durante el desarrollo de la aplicación, se han utilizado las librerías de acceso libre de *rosjava*, las cuales permiten utilizar métodos que combinan el código en lenguaje *Java* con el sistema de ROS como base de la *app*.

Para el correcto funcionamiento de la aplicación es necesario:

- Tener un PC y un dispositivo móvil o *tablet* conectados a la misma red de Internet.
- Mantener el *master* de ROS en ejecución asignando valor a las variables *ROS_MASTER_URI* y *ROS_IP*. En estas direcciones asignamos la dirección IP (*Internet Protocol*) del ordenador utilizado y la dirección IP a la que se conectará el dispositivo móvil. Es posible asignar un valor fijo a estas variables, ya que ROS dispone de un fichero en el que almacenar las variables de entorno, lo cual puede ser más cómodo siempre que se trabaje con la misma red de Internet. En caso contrario, habrá que asignar un valor antes de ejecutar el *master*.
- Existencia de un nodo de ROS que esté publicando información en *topics*.

En caso de que alguna de estas condiciones no se cumpla ocurrirá lo siguiente:

- Si alguna de las direcciones comentadas no es correcta, que en este caso será la misma para las dos variables, la aplicación advertirá al usuario de la imposibilidad de conexión al PC. Esto también puede ocurrir si no se está corriendo el *master* de ROS.
- Si no hay ningún nodo de ROS publicando información, la aplicación no mostrará información en su interfaz.
- La aplicación no funcionará correctamente en caso de que alguno de los dos dispositivos tenga problemas de conexión a la red, o no estén conectados a la misma.

Por tanto, la aplicación comienza mostrando una pantalla en la que el usuario debe ingresar la dirección correspondiente a la variable *ROS_MASTER_URI*. Como se puede observar en el diagrama del anterior subcapítulo, este paso corresponde al bloque "*Connect"*.

- **Explicación del desarrollo:**

Para la obtención de este resultado, se utiliza una clase ya existente, llamada *MasterChooser Class.* Esta clase es proporcionada por los paquetes base incluidos en la instalación del propio sistema *Android Studio*.

El funcionamiento correcto de este primer paso se consigue añadiendo la dirección en red que tiene la clase utilizada al fichero *Android Manifest* del proyecto creado para la aplicación. En él, se establece que esa clase debe ser la ejecutada cuando se arranque la aplicación en el dispositivo móvil y los permisos de acceso a red necesarios para la conexión entre los dispositivos móvil y PC.

El fichero *Android Manifest* es un archivo de configuración de los proyectos de *Android* donde se establecen las preferencias de la aplicación, los permisos de acceso en red si fuesen necesarios, y describe los componentes necesarios utilizados en la *app*, como clases, estructuras y actividades. También se establecen las versiones mínimas y máximas de Software que son capaces de soportar dicha aplicación.

Es necesario comentar, que la aplicación sigue una estructura por fragmentos o *fragments*[12], la cual permite cambiar de manera rápida y sencilla la visualización mostrada por pantalla para una misma *Activity*, en nuestro caso: *Main Activity* [28].

Tras este paso, la aplicación ya se ha conectado al PC en cuestión, y muestra una primera pantalla en blanco, sin información, con un botón en la esquina superior derecha de la vista con el símbolo de "Actualizar" llamado "Botón". Esta visualización pertenece a la vista del "*List Fragment* vacío" del diagrama. En el momento en que el botón es pulsado, la aplicación realiza lo siguiente:

- *List Fragment* cuenta con un método para mostrar la información publicada por el nodo de ROS en ejecución filtrando dos tipos: *Compressed Images* y *PointCloud2*.

- Éste método llama a un método de la actividad y ésta a su vez, hace el mismo proceso con una clase llamada *ROS Node*. Esta clase contiene:

    - Los pasos necesarios para crear un nodo que se subscriba a los *topics* publicados en el PC.
    - Un método para obtener la información de nombre y tipo de los datos que se quiere mostrar en la aplicación.
    - Además, contiene métodos para obtener los datos, imágenes o nubes de puntos que posteriormente se muestren por pantalla.
    - Por último, también cuenta con un método para eliminar el nodo creado y que no se mantenga activo cuando no sea necesario.

Tras esto, en la pantalla aparece un listado de *topics* de los tipos requeridos, correspondiente en el diagrama al bloque "*List Fragment* lleno".

- **Explicación del desarrollo:**

A través del fragmento donde se listan los elementos, se llama al método *getTopics* de la clase principal *Main Activity*. Esto es posible gracias a la reutilización de código que permite la programación orientada a objetos.

En el método *getTopics* de la clase principal se llama al método llamado de la misma manera perteneciente a la clase *Ros Node*, que accede, mediante la conexión establecida al arranque de la aplicación, a los *topics* publicados y filtra los que sean de tipo *Compressed Image* y *PointCloud2*.

[12] *Fragments:* representan el comportamiento de una parte de un conjunto de ellas, las cuales forman en su totalidad la actividad. Se considera una sección modular de la actividad que cuenta con ciclo de vida propio.

Tras esto, desde el método de *Ros Node*, se devuelve el dato y ocurre lo mismo de la clase principal, hasta devolver la información al fragmento.

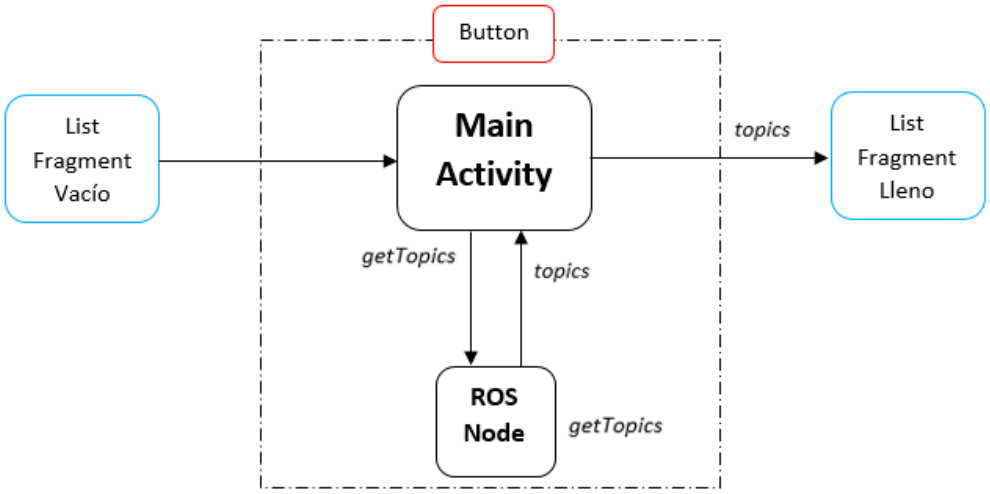

Fig. 4.5. Diagrama del flujo de *topics* desde el PC a la pantalla de la aplicación.

Se puede acceder a la información de estos *topics* pulsando sobre ellos. De este modo, se hace una llamada a la actividad para que cambie la visualización mostrada por pantalla:

- Si el *topic* sobre el que se pulsa es de tipo *Compressed Image*, se realiza la llamada a un método de la actividad y a su vez, la actividad llama al método correspondiente contenido en la clase *ROS Node*. En este caso, en la clase se crea un *Subscriber* y se accede a la información del *topic* diferenciado por su nombre y tipo. En el método correspondiente se realiza un mapeo de la información y se procesa convirtiendo los datos en un tipo que pueda ser procesado para mostrar la imagen.

- Si el *topic* del que se requiere la información es de tipo *PointCloud2*, el funcionamiento del código es el mismo a través de otro fragmento llamado "*PointCloud Fragment*".

- **Explicación del desarrollo:**

En el momento en que un *topic* de la lista es seleccionado, la funcionalidad de la aplicación se establece a través de un método llamado *setOnItemClickListener*, con el cual se llama al método *showFragment* de la clase *Main Activity.*

En el método mencionado de la clase principal, *showFragment*, se implementa la decisión de qué diseño de pantalla abrir, eligiendo entre el *Viewer Fragment* o el *PCL Fragment*. Esta funcionalidad se obtiene a través de sentencias condicionales *if* que cubren tres posibilidades: que la selección sea una imagen comprimida, que sea una nube de puntos, o que no haya *topics* seleccionados. Este último caso se implementa para mantener la *app* en funcionamiento y evitar fallos de compilación si no hay *topics* publicados en el momento de la selección.

- Selección *Compressed Image:*

Si el *topic* seleccionado es de imagen, entonces el método *showFragment* abrirá el diseño del *Viewer Fragment*. En el momento de la creación de esta pantalla se ejecuta un método que sigue la línea de ejecución mostrada en la siguiente figura.

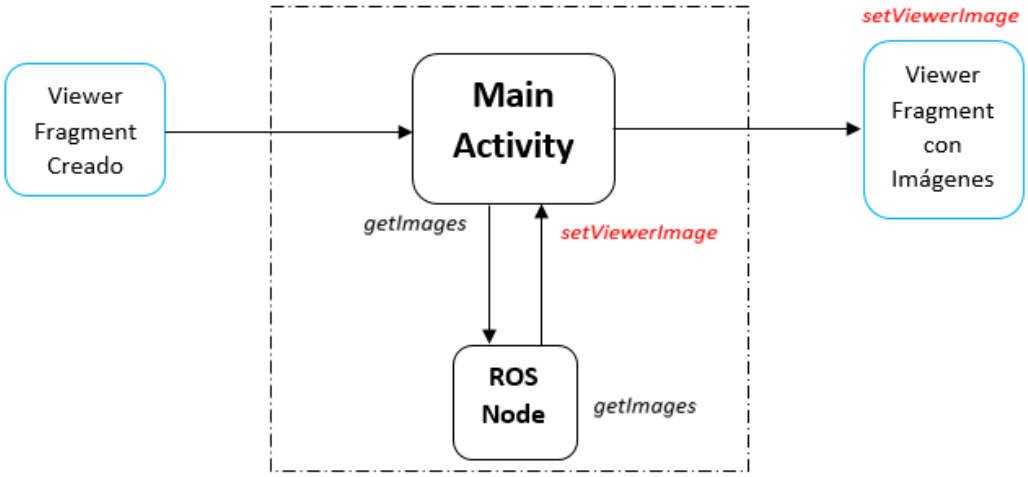

Fig. 4.6. Diagrama del flujo de imágenes recibidas en la aplicación.

Como se observa en el diagrama, el método *getImages* de la clase principal funciona de manera análoga al método *getTopics* anteriormente explicado: realiza una llamada al método *getImages* perteneciente a la clase donde se encuentra el nodo activo, la clase *Ros Node*.

A continuación, en ese método se realiza la inicialización del *Subscriber*, declarado como variable global, asignándole el tipo de dato que se quiere obtener. A este objeto se le añade la funcionalidad de un bucle para obtener imágenes cada vez que se publique una nueva.

Para el procesado de imágenes, se crea un objeto del tipo *Bitmap*[13] y se ejecuta un método llamado *decodeImageMessage*, obtenido de las librerías del tipo comentado, que decodificará el mensaje obtenido en un tipo de dato compatible para ser procesado.

Por último, después de ser procesado, se envía la información llamando al método *setViewerImage* de la clase principal. Este método realiza una llamada al mismo perteneciente al *Fragment Viewer*, que se encarga de mostrar la imagen obtenida.

- Selección *PointCloud2:*

Por otro lado, si la selección del listado pertenece a un *topic*, cuya información publicada es de tipo *PointCloud2*, el método *showFragment* ejecutará el *PCL Fragment* siguiendo la misma metodología que para el caso anterior.

Siguiendo la misma arquitectura, se realiza la llamada *getPCD* de la clase principal en el momento de la creación del fragmento y desde el método de la *Main Activity* se llama al *getPCD* correspondiente a la clase *Ros Node*.

[13] *Bitmap* es un formato de imagen de mapeo en bits.

En este método se realiza la inicialización del *Subscriber* de forma que el tipo de dato a identificar sea *PointCloud2*. Sin embargo, en este caso, se obtiene el mensaje y se devuelve hacia la clase principal a través de la llamada al método *setPointcloud*. Desde ahí, la información pasa al fragmento correspondiente llamando al método *setPclViewer* del mismo, donde se procesará y se mostrará la nube de puntos resultante.

El diagrama que sigue este funcionamiento es el siguiente:

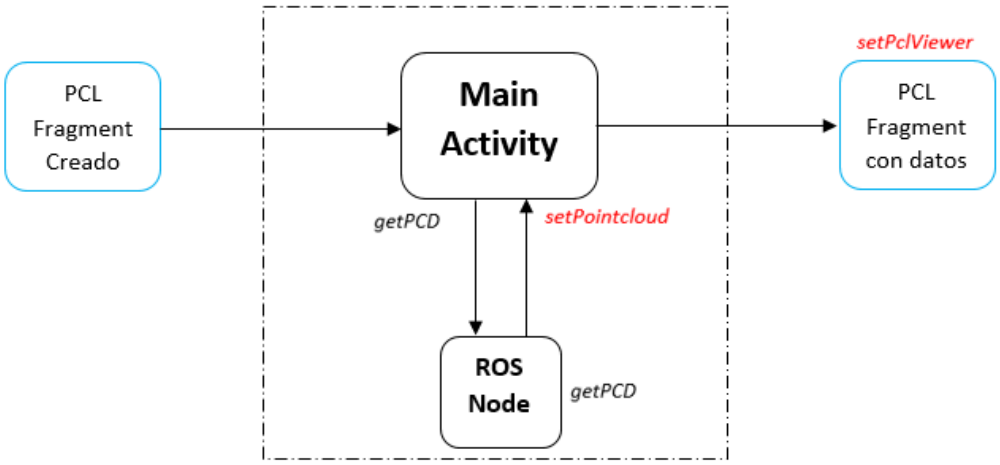

Fig. 4.7. Diagrama explicativo de la recopilación de datos *PCL*

Para obtener la nube de puntos en la aplicación, es necesario procesar los datos obtenidos desde la clase *Ros Node*. Para este objetivo, se crean cuatro clases obtenidas de las librerías *OpenGL*, ajustadas y modificadas para que realice las acciones necesarias.

El diagrama que sigue este proceso es el siguiente:

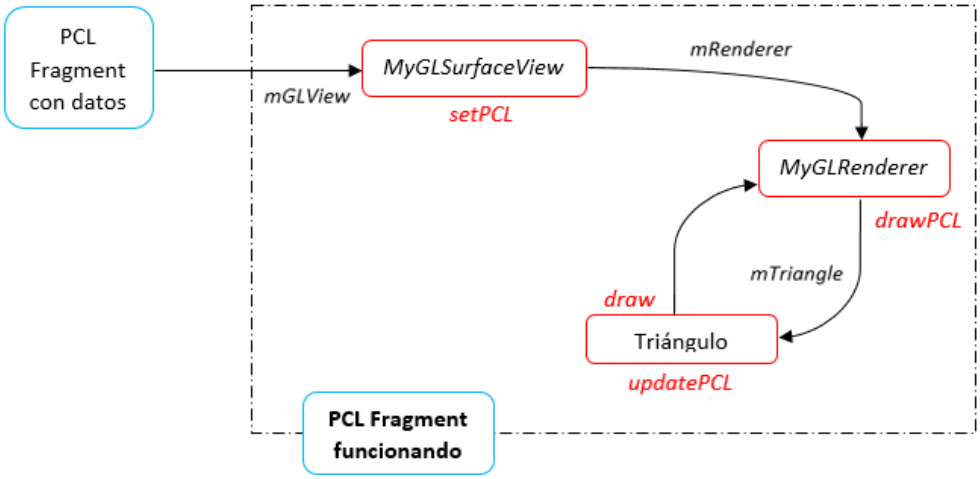

Fig. 4.8. Flujo de datos para la impresión de las nubes de puntos.

Como se puede observar en el diagrama, la secuencia para obtener los datos se lleva a cabo desde el fragment, con la información en bruto, y pasando por diferentes clases que van obteniendo la información pertinente. Con la clase *MyGLSurfaceView* se llama a un método de la clase *MyGLRenderer (drawPCL)*, que se encarga de llamar a los métodos de dibujo y actualización de la nube de puntos pertenecientes a la clase "Triángulo".

Una vez obtenida la nube, y teniendo activa la actualización con el método *updatePCL*, se obtienen los resultados impresos por pantalla de manera continua.

Ambos *fragments* implementados cuentan con un botón llamado "Volver" diseñado en la esquina superior derecha de la pantalla de visualización, cuya funcionalidad es la de volver al inicio de la aplicación, mostrando el *layout* correspondiente al llamado "*List Fragment* vacío".

4.2.4. Otros modelos implementados

Durante el periodo de desarrollo e implementación de la aplicación, se partió de una idea base, que se fue modificando hasta llegar al resultado del modelo explicado en el anterior apartado. A continuación, se explican los modelos implementados previos a la estructura final y los motivos por los cuales no se tomaron como modelos definitivos:

4.2.4.1. Modelo de actividades múltiples

La primera idea de desarrollo de la aplicación consistía en el uso de tres actividades:

a. La actividad principal de la aplicación: En esta actividad es en la que se mostraría un listado vacío de *topics* de manera inicial, de la misma forma que el modelo resultante. Esta actividad principal fue implementada con extensión de ROS, y su funcionalidad sería:

- Creación y ejecución de un nodo de ROS a través del cual obtener la información de los *topics* publicados en el PC.
- Creación de un método que obtenga los *topics* en cuestión de los tipos deseados (*Compressed Image* y *PointCloud2*) desde el PC y los publique en forma de listado.

- Mediante la arquitectura de mensajes *Intent*[14] se pasa la información del *topic* seleccionado, desde la actividad principal hasta la actividad correspondiente al tipo de *topic* seleccionado.

b. Actividades de visualización: Dependiendo del tipo de dato seleccionado, el *topic* se transfería a una actividad u otra. La funcionalidad de ambas sería la siguiente:

- Obtención del nombre del *topic* seleccionado a través de la estructura de mensajes mencionada anteriormente.

- Procesado y transformación del tipo de dato publicado por el *topic* y posterior impresión por pantalla.

- Además, ambas actividades contaban con un botón en la esquina superior derecha de la pantalla, cuya funcionalidad consiste en volver a la actividad principal.

[14] Es la estructura más habitual en *Android Studio* de transmisión de información y lanzamiento de nuevas actividades de una aplicación en ejecución.

Durante la depuración del modelo, se observó que, en el proceso de transmisión de la información desde la actividad principal a cualquiera de las actividades de visualización de resultados, la aplicación mostraba un comportamiento anómalo y se forzaba el cierre de la ejecución.

Para ir comprobando cuál era el problema que causaba dicho comportamiento, se fue retrocediendo hasta una aplicación con la actividad principal operativa, y una actividad nueva vacía, sin funcionalidad. Con esto, se comprobaba que el lanzamiento de la nueva actividad era correcto. En ese momento se comprobó que el error nacía ahí.

Junto con la actividad principal, el nodo de ROS actuaba de la misma manera, y se mantenía en ejecución en segundo plano en el momento de lanzamiento de la segunda actividad (visualización de *topics*). Al hacerlo, se ejecutaba de nuevo y de manera automática la creación de un nodo de ROS.

Este funcionamiento podría considerarse algo lógico, normal. Pero, se pudo comprobar a través de los foros de *Android* y ROS que, las versiones instaladas de *Android Studio*, *Java* y *Ubuntu Linux*, no permitían el uso de dos nodos de ROS de manera simultánea. Esto fue lo que ocasionó la implementación del siguiente modelo.

4.2.4.2. Modelo de actividades múltiples con nodo encapsulado

La siguiente versión de aplicación pensada fue una variación del anterior modelo expuesto:

- Funcionalidad de las actividades idéntica al modelo descrito anteriormente.

- Para evitar problemas en la aplicación de lanzamiento y detención del nodo creado para la ejecución del proceso habitual, se creó una clase en la cual se aplicaba la extensión de ROS. Con esto, se mantendría el nodo activo a pesar del cambio de actividades producido durante la transmisión de mensajes.

Con la finalidad de depurar el error observado en el primer modelo propuesto, se creó una clase llamada *RosNode*. Con ello, se evitaría la creación de dos nodos de ROS de manera simultánea, sino que se mantendría ajeno al cambio de actividades.

En esta ocasión, se observa un comportamiento parecido al obtenido en el modelo anterior. Dado que las actividades de visualización requieren de la extensión de ROS para obtener los datos publicados en el *topic* seleccionado e imprimirlos por pantalla, la aplicación seguía generando un error.

El error obtenido se producía en el momento de lanzamiento de la segunda actividad: al ser una actividad con extensión ROS, el *Android Manifest* interpretaba la necesidad de conectar el nodo con el PC, por lo que intentaba lanzar la clase de conexión *MasterChooser*. El resultado, de nuevo erróneo, forzaba el cierre de la aplicación en el momento de seleccionar uno de los *topics* listados.

4.2.5. Ventajas e inconvenientes del modelo propuesto

En este apartado se explican los motivos por los cuales se decide utilizar el modelo explicado en el subcapítulo 4.2.3. y las ventajas e inconvenientes de utilizar dicho modelo.

- **Ventajas:**

Con la clase *Ros Node* implementada, y una única actividad con extensión ROS, se evitan los problemas causados en los anteriores modelos y se comprueba durante las depuraciones pertinentes la correcta funcionalidad de la *app* sin cierres forzados de manera inesperada e irregular.

Por tanto, con los *fragments* se mantiene un único nodo de ROS en ejecución, que, como se comentaba anteriormente, era uno de los problemas de ejecución. Además, los *fragments* permiten cambiar de pantalla de manera rápida y sencilla, sin problemas de consumos graves de la batería del dispositivo utilizado.

- **Inconvenientes:**

La decisión de realizar el modelo resultante con una única actividad conllevó el aprendizaje de trabajo con las estructuras de tipo *Fragment* y el cambio de toda la funcionalidad de la aplicación desde el nivel de diseño más bajo.

Por otro lado, la idea inicial consistió en obtener de manera visual la información relacionada con nubes de puntos e imágenes. El principal inconveniente de utilizar fragmentos es la cantidad de pasos intermedios que se deben dar, desde el momento en que se selecciona un *topic* determinado hasta la visualización del mismo.

En particular, el principal problema se obtiene con los objetos de tipo *Image*, debido a que el procesado de la misma se realiza pixel a pixel y las imágenes son de una calidad suficiente como para hacer de este proceso un trabajo largo y pesado.

Si ese problema, se suma a la cantidad de pasos que se realizan para pasar la información de cada pixel desde la clase *Ros Node*, donde se procesan los datos, hasta la clase "*Fragment Viewer*", se obtiene un resultado poco óptimo.

Por tanto, con el fin de obtener resultados más rápidos, se cambió la funcionalidad de la aplicación realizando el procesado de imágenes de tipo *Compressed Image*. En este caso, el procesado se realiza casi inmediato y el traspaso de información entre clases no supone un problema añadido, a diferencia del caso anterior.

5. RESULTADOS EXPERIMENTALES

En este capítulo de la memoria, se exponen ejemplos de ejecución de ambas aplicaciones, cuya funcionalidad ha sido comentada de manera detallada en el capítulo inmediatamente anterior.

Para el funcionamiento correcto de ambos procesos es necesario realizar los siguientes pasos:

I. Abrir dos terminales independientes en el SO *Ubuntu Linux* utilizado.
II. Ejecutar el comando "*ifconfig*" para obtener la dirección IP del ordenador.
III. Comprobar la conexión de PC y dispositivo móvil a la misma red de Internet.
IV. Establecer el valor de las variables de entorno "*ROS_MASTER_URI"* y "*ROS_IP"* con el valor obtenido del comando del paso II.
V. Ejecutar el comando *"roscore"* que lanza el *master* de ROS.

5.1. Demostración de funcionamiento del Visualizador

Tras los primeros pasos mencionados anteriormente, se puede comenzar el proceso de compilación del visualizador de nubes de puntos. Para comprobar dicho funcionamiento se realiza lo siguiente:

- Abrir un terminal vacío y asignar de nuevo las variables de entorno anteriores con el valor correspondiente a la dirección IP, y actualizar el archivo contenedor de las variables de entorno.

- Ejecutar con el comando "*rosbag play*", el cual compila archivos con estructura de ROS, seguido de la dirección de la carpeta donde se encuentra el fichero y el nombre del mismo. Con esto, comienza la ejecución del archivo que contiene la información captada por el vehículo durante la grabación del *bag*.

- En otro terminal nuevo, se ejecuta el archivo de *Qt Creator* con extensión *"cpp"* creado para el visualizador.

El resultado es el siguiente:

Fig. 5.1. Estructura de compilación del visualizador.

El funcionamiento del visualizador, como se explica en el apartado 4.1., está implementado de manera que, cada vez que se publique en el *topic* de tipo *PointCloud2* un dato nuevo, éste cambie la nube de puntos mostrada por la nueva obtenida. En la parte llamada "código implementado" se muestra un mensaje cada vez que llega una *pcd* nueva.

A continuación, se muestra una secuencia de funcionamiento donde se aprecia el que la nube de puntos evoluciona con el tiempo:

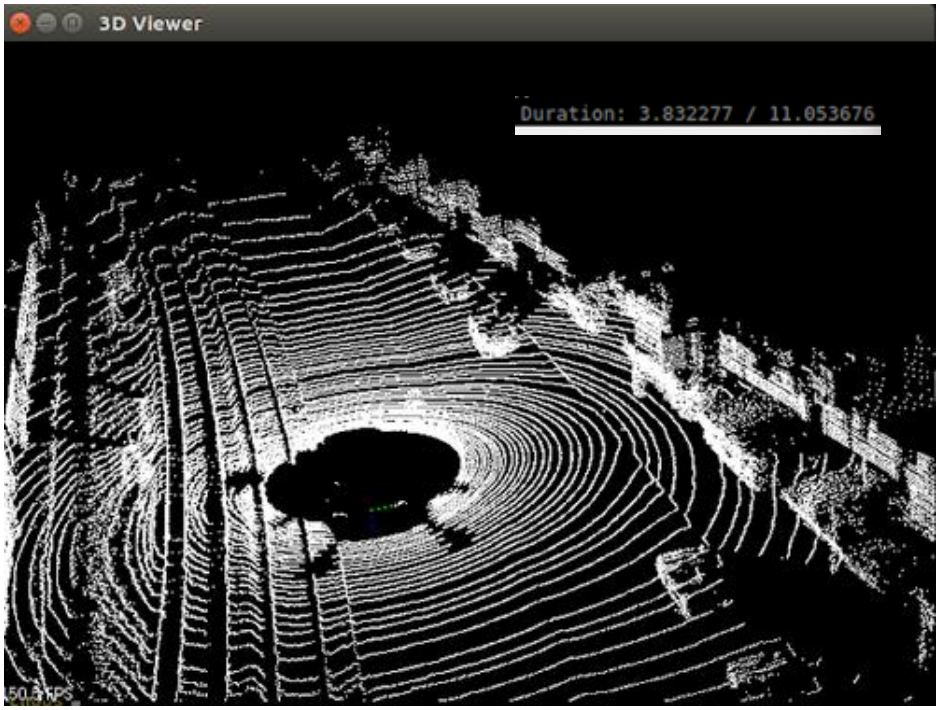

Fig. 5.2. Visualizador de *Point Clouds.* Tiempo de ejecución: 3.83s

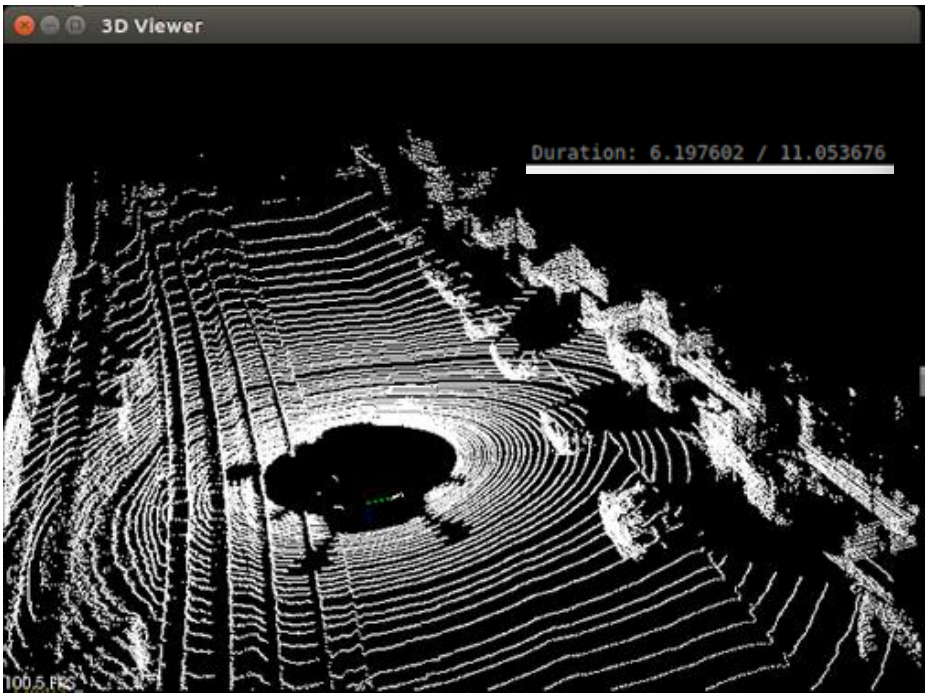

Fig. 5.3. Visualizador de *Point Clouds*. Tiempo de ejecución: 6.20s

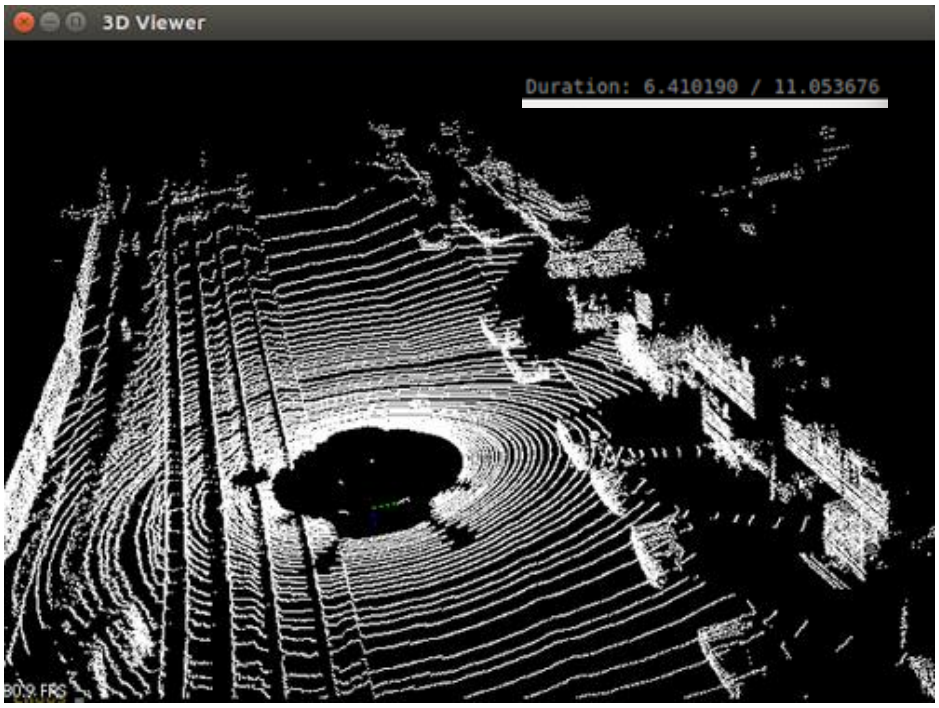

Fig. 5.4. Visualizador de *Point Clouds*: Tiempo de ejecución: 6.41s

Como se puede observar en las imágenes, se ha insertado un extracto del temporizador en cada una de ellas con el objetivo de comprobar la diferencia entre las nubes de puntos de las tres imágenes expuestas.

Entre las figuras 5.2. y 5.3., hay un margen de aproximadamente 3 segundos y se puede apreciar en el lateral derecho de la carretera, la diferencia de los vehículos aparcados.

Por otro lado, entre las figuras 5.3. y 5.4., la diferencia de tiempo es muy pequeña, pero en la esquina superior derecha de la última, se puede comprobar una detección más lejana que no aparece en la anterior.

Este visualizador, tiene la capacidad de ser ajustado para un mejor análisis, según determinados comandos de ratón y teclado del ordenador. Estos comandos, permiten movimientos de zoom y rotación sobre cada uno de los ejes.

En total, es posible realizar cinco movimientos diferentes utilizando patrones de movimiento del ratón:

- Zoom:
    - Si el zoom se quiere aplicar a través del teclado táctil de un ordenador portátil, el movimiento a realizar con los dedos es el que se puede apreciar en la siguiente imagen.
    - Si se utiliza un ratón periférico, el movimiento de zoom se realiza con botón derecho + movimiento hacia delante.

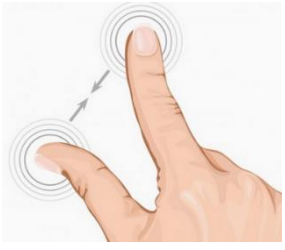
Fig. 5.5. Gesto de zoom

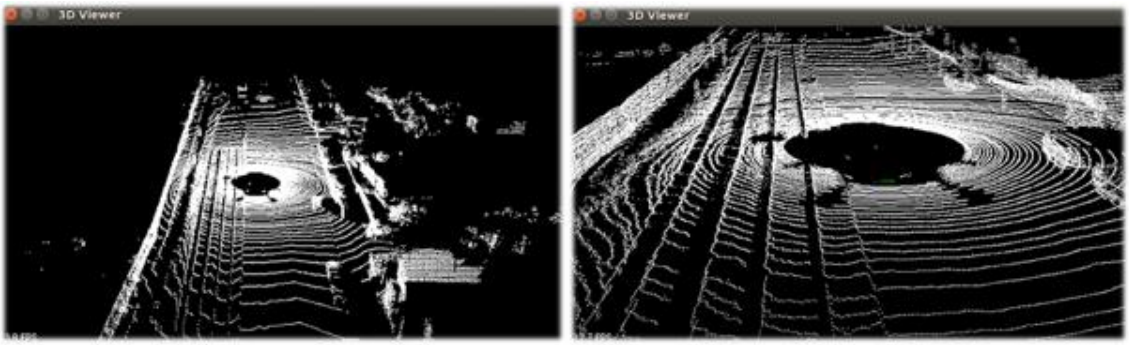

Fig. 5.6. Vista del visualizador con diferentes escalas de zoom

- Rotación:
    - Por otro lado, la rotación respecto al eje vertical se realiza con el botón izquierdo + movimiento hacia los lados. Este caso representa el eje dibujado de rojo en la en la siguiente figura (Fig. 5.7.).
    - Con botón izquierdo + movimiento hacia delante o detrás se obtiene rotación respecto al eje horizontal paralelo a la pantalla del ordenador, el cual aparece representado en color verde.
    - Por último, la rotación respecto al eje que sale perpendicular a la pantalla, color amarillo de la imagen, se obtiene con el comando *Ctrl* + botón izquierdo + movimiento hacia los lados.

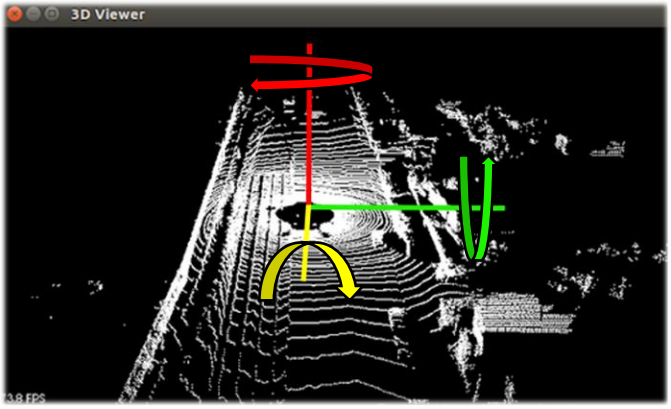

Fig. 5.7. Ejes que guían las posibles rotaciones explicadas

- Desplazamiento: con el comando *Shift* + botón izquierdo + movimiento hacia los lados se obtiene desplazamiento paralelo a la pantalla (arriba, abajo, izquierda y derecha).

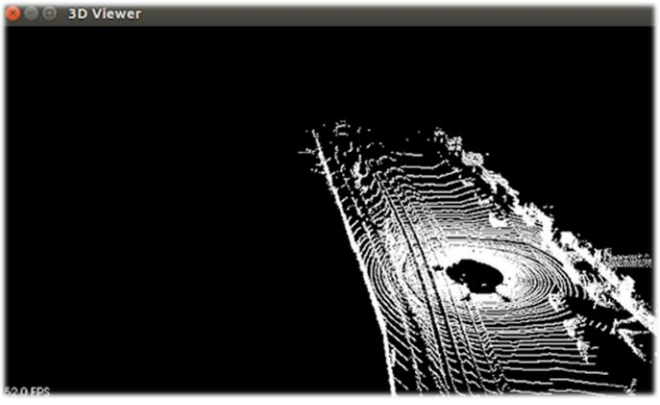

Fig. 5.8. Vista del visualizador desplazada

5.2. Demostración de funcionamiento de la *app LSI Play Remote*

Después de realizar los pasos I-V descritos en el comienzo del capítulo 5., lo siguiente que hay que hacer para el funcionamiento de la *LSI Play Remote app* es:

- Abrir un terminal en el que establecer de nuevo los valores de las variables de entorno *"ROS_MASTER_URI"* y *"ROS_IP"* en función de la dirección IP registrada en el PC.

- Posteriormente, ejecutar un fichero con extensión *bag* compatible con la arquitectura de ROS que contenga *topics* con datos recogidos de los tipos *Compressed Image* y *Pointcloud2*, o al menos, uno de los dos, para poder comprobar la correcta funcionalidad de la aplicación.

- Por último, es necesario disponer de un dispositivo móvil que cumpla las condiciones de versiones descritas en el *Android Manifest* o un emulador configurado para tal efecto. Ejecutar la aplicación en el dispositivo y comienza la ejecución abriéndose una interfaz donde pide insertar la dirección IP del PC al que se va a conectar.

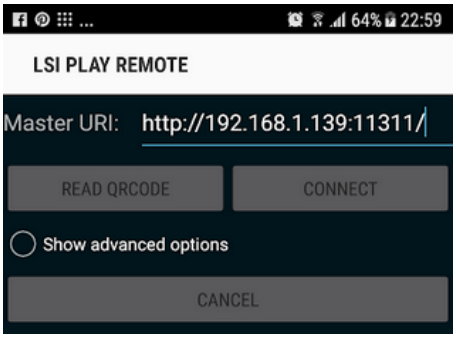

Fig. 5.9. Vista de la clase *MasterChooser*

Al insertar la dirección IP pueden ocurrir dos cosas: la dirección IP no es correcta, y la aplicación muestra un mensaje emergente avisando de la imposibilidad de conexión con el PC, o la dirección IP es correcta y la aplicación se inicia de manera correcta. Se muestran ambos casos en las siguientes figuras:

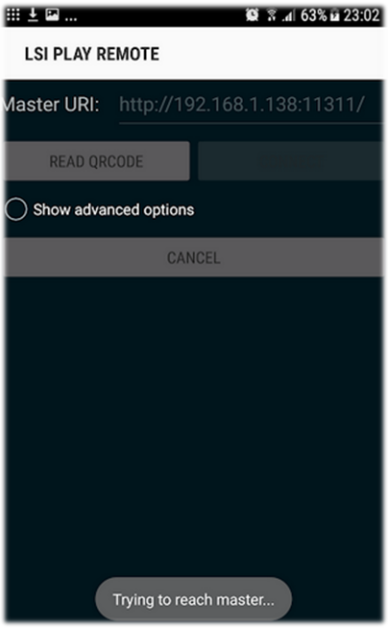

Fig. 5.10. Intento de arranque con dirección IP errónea.

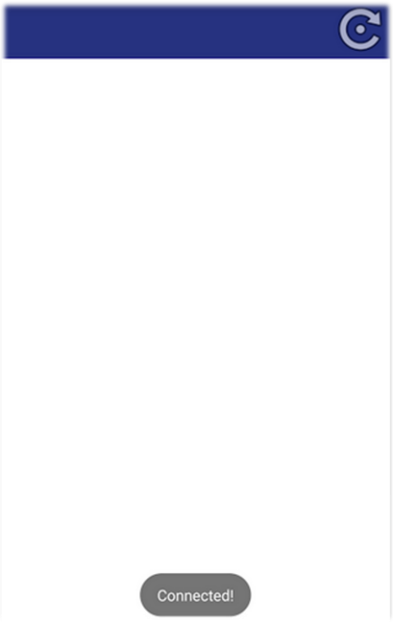

Fig. 5.11. Arranque de la *app* con dirección IP correcta.

Después de ingresar la IP correctamente y arrancar por completo la aplicación, se puede comenzar con la visualización de los *topics*. Pulsando el botón de la esquina superior derecha que se ve en la siguiente imagen, se obtiene un listado de *topics* de los tipos filtrados en el código: *Compressed Image* y *Pointcloud2.*

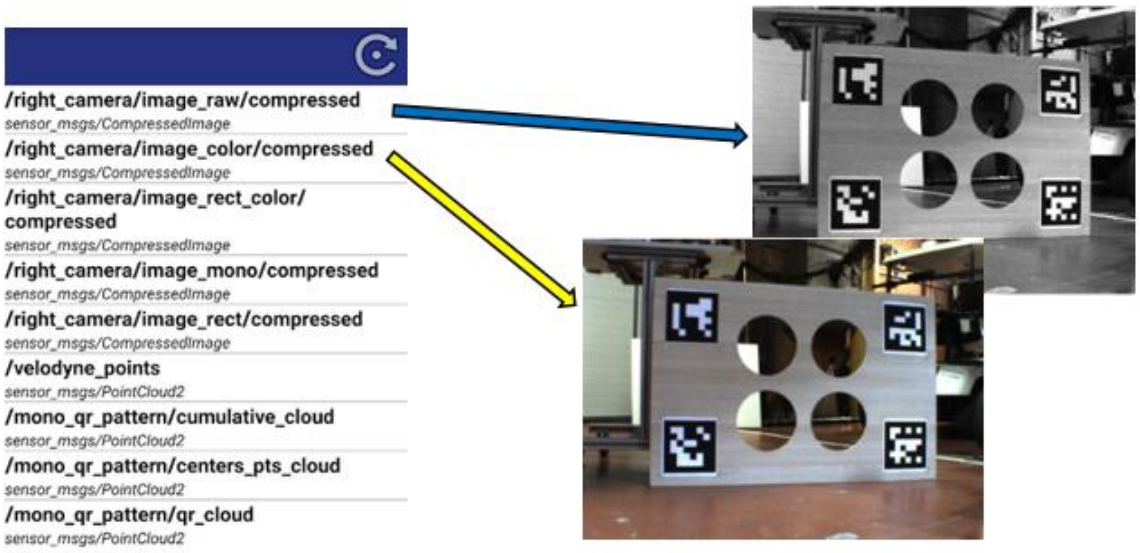

Fig. 5.12. Vista del listado. Visualización de imágenes a color y en escala de grises.

Como se puede observar en la imagen anterior, el listado que aparece, muestra en dos líneas la información de los *topics*. Dicha información corresponde al nombre, en la primera línea, y al tipo de *topic*, en la segunda línea.

A partir de aquí, y haciendo mención al diagrama de bloques de la figura 4.4., en función de la selección, la aplicación actúa de diferente forma para mostrar la información solicitada:

- Pulsando sobre un elemento de la lista, cuyo tipo de información contenida es *Compressed Image*, se abre el visualizador de imágenes. Dado que las cámaras captan imágenes de manera continua, la actualización, que permite observar la última imagen captada, simula la ejecución de un vídeo. Este efecto se puede comprobar en las siguientes imágenes.

Fig. 5.13. Visualización de *topic Image* 1.

Fig. 5.14. Visualización de *topic Image* 2.

Fig. 5.15. Visualización de *topic Image* 3.

Fig. 5.16. Visualización de *topic Image* 4.

Es posible volver a la pantalla principal presionando sobre el botón que aparece en la interfaz, situado en la esquina superior derecha.

- Por otro lado, en caso de pulsar sobre un elemento de tipo *PCL*, la aplicación abrirá el visualizador de nubes de puntos implementado en el *PCL Fragment*. Dicho visualizador también incorpora un botón de retroceso en la esquina superior derecha.

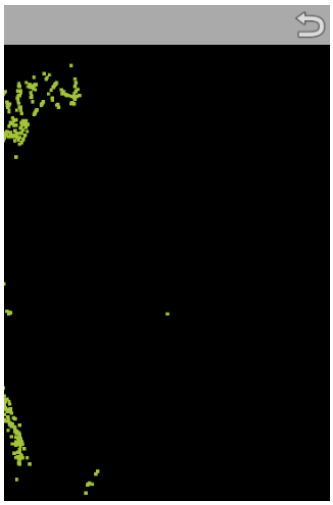

Fig. 5.17. Visualización de *topic* PCL 1

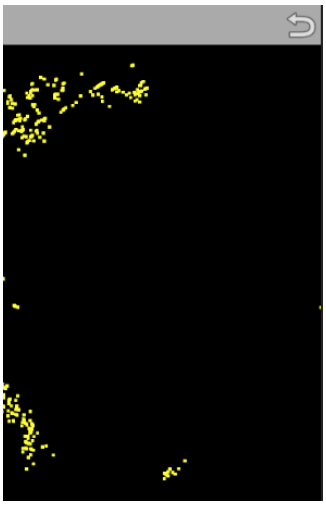

Fig. 5.18. Visualización de *topic* PCL 2

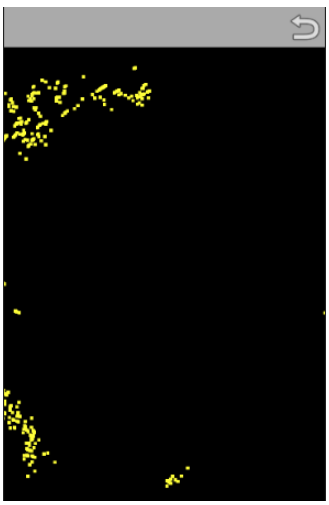

Fig. 5.19. Visualización de *topic* PCL 3

Como se puede observar en las tres imágenes mostradas, la visualización es muy parecida, y es debido a que el archivo *bag* utilizado para reproducir la aplicación, consistía en la captación de un entorno encontrándose el vehículo inmóvil. La diferencia principal se encuentra en la primera, donde aparece el centro de coordenadas del PCL mostrado.

Utilizando la aplicación *Rviz*, con la que se puede visualizar el PCL desde un PC, se puede comprobar el óptimo resultado obtenido en la aplicación.

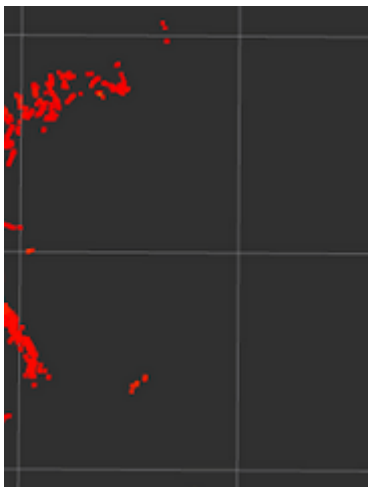

Fig. 5.20. Visualización de *topic* PCL con *Rviz*

Además, se ha implementado una función sobre el botón que incorporan los *smartphones* para volver a la pantalla anterior, del siguiente modo:

- Si se pulsa una vez en el botón, aparece una ventana emergente que proporciona la posibilidad de salir de la *app* si vuelves a pulsar sobre el botón en un espacio de tiempo reducido con respecto al primer toque.
- Si se pulsa la segunda vez pasado un tiempo de espera programado en el código, volverá a mostrarse el mensaje mencionado.

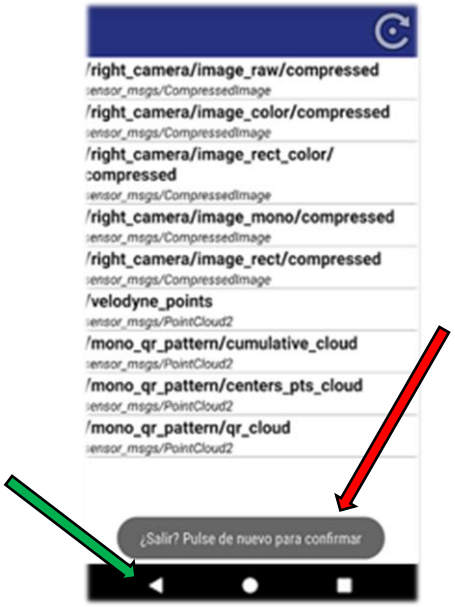

Fig. 5.21. Ejemplo de salida de la aplicación.

6. TRABAJOS FUTUROS

En este capítulo se expondrán ideas de posibles trabajos futuros y mejoras que se puedan realizar sobre las implementaciones creadas en el presente proyecto.

6.1. Mejoras sobre el visualizador

Una de las mejoras que se pueden llevar a cabo en el visualizador, consiste en añadir un menú lateral o en la zona inferior del visualizador que permita cambiar los colores siguiendo la escala de colores *RGB* (*Red-Green-Blue*), de manera que aporte mayor calidad a la nube de puntos representada.

Para ello, es necesario seguir unos pasos proporcionados por la web de nubes de puntos comentada en la bibliografía. Uno de los pasos, es el de obtener e instalar el compilador *cmake*, el cual permite trabajar con la mejora comentada.

Además, podría ser muy útil añadir a ese menú botones para realizar desplazamientos en la visualización, como el zoom, la rotación que permite hacer con el ratón respecto a los ejes o movimientos de un lado a otro de la nube.

Otra mejora posible referente al color, podría ser la implementación de métodos que pinten determinados grupos de vértices del entorno de un color determinado. De esta forma, al encontrar grandes agrupaciones de vértices en zonas concretas y reproducirlas con un color determinado, diferente al de otros grupos de alrededor, permitiría obtener la identificación de diferentes objetos.

Realizar modificaciones en el visualizador, de manera que sea capaz de procesar nubes de puntos que aporten más datos como la textura, podría ser una mejora considerable.

6.2. Mejoras sobre la aplicación de móvil

Por otro lado, como propuesta de mejora es posible realizar cambios sobre el código para que el listado de *topics* aparezca de forma automática después de arrancar con la conexión al *master*.

Para ello, habría que eliminar el botón implementado y llamar desde la creación del *fragment* correspondiente al método que devuelve los *topics*. Esto podría durar unos segundos debido al traslado de información que pasa por la actividad principal y la clase *Ros Node* contenedora de los métodos para la obtención de ese listado. Por tanto, al arrancar, es posible que durante 2 o 3 segundos, dependiendo también de la calidad de la señal de Internet, el listado *fragment* aparezca vacío. Además, el proceso de volver al listado después de visualizar alguna imagen o nube de puntos sería parecido e igual de lento. Pero, por otro lado, no es un problema de gravedad que reste funcionalidad a la aplicación.

Por otro lado, sería interesante implementar el código de manera que permita la rotación de pantalla de manera automática durante la ejecución. Esto, en una aplicación

sin extensiones de ROS puede ser automático añadiendo una línea al código de diseño correspondiente a la actividad. Sin embargo, por lo que se ha podido comprobar, para aplicaciones con ROS no es tan sencillo. Además, esto sí podría aportar una ralentización considerable sobre el funcionamiento de la aplicación. Una opción para realizar esto, sería crear dos fragmentos para cada uno de los existentes actualmente, es decir, dos para el listado, dos para el visualizador de imágenes, y otros dos para el visualizador de nubes de puntos. De esta forma, detectando por medio de los sensores incorporados en el teléfono o *tablet* la posición en la que se encuentra, se ejecutaría un fragmento cuyo diseño fuese apaisado, o el actual.

También se podría añadir funcionalidad al visualizador de la nube de puntos, dotándolo de botones para actuar sobre la orientación, distancia y enfoque de la cámara a través de la cual se ve la nube. Aunque esto disminuiría el espacio disponible para la visualización, lo cual podría resultar contraproducente, en caso de que la aplicación se ejecute en un teléfono móvil con una pantalla de 4 o 5 pulgadas (tamaño habitual actual).

Otra posible mejora aplicable, consistiría en implementar la aplicación creada en el sistema *Mac OS* del fabricante *Apple*. Esto proporcionaría mayor soporte, ya que la aplicación sería compatible con los dos sistemas operativos líderes en el sector de la telefonía.

Por último, se podría estudiar la manera de aumentar la velocidad de procesado en todo tipo de imágenes, para poder visualizar no sólo imágenes de tipo *Compressed Image* y otro tipo de datos publicados por los *topics* que aporten información relevante a los ocupantes del vehículo: datos de GPS, odometría o marcadores geométricos para mostrar la diferenciación obtenida de los objetos detectados.

A nivel de depuración, sería interesante poder mejorar el espacio requerido de memoria RAM (*Random Access Memory*) durante la ejecución de la aplicación, y también se podría estudiar el trabajo de la aplicación con el fin de mejorar el consumo de batería durante su ejecución.

7. MARCO REGULADOR

A continuación, se expone en este capítulo la legislación actual aplicable al ámbito que engloba los sistemas de percepción, el vehículo autónomo y el desarrollo Software.

7.1. Legislación vigente en el vehículo autónomo

Aunque uno de los problemas de la no regulación del vehículo autónomo actualmente deriva de los problemas conocidos y limitaciones comentadas en el capítulo 2, muchos científicos y desarrolladores involucrados en el estudio del vehículo autónomo consideran que los factores políticos, jurídicos, de regulación, infraestructura y responsabilidad son temas que también hay que estudiar para la regulación, como afirma Noelia López en el artículo publicado en la revista virtual llamada "*Autobild*" [29].

- **América:**

En Estados Unidos, se trabaja en la actualidad para obtener los permisos y establecer las normas que regulen esta situación, con el fin de permitir a los vehículos autónomos circular por las calles con tráfico abierto. Esto es debido a que muchos estados ya tienen legislación que permita esto, pero, para regularlo de manera oficial a nivel nacional, es necesario que todos los estados regularicen esta situación.

No obstante, actualmente los estados que poseen normas que regulen la conducción autónoma, las tienen vigentes únicamente para organizaciones y fabricantes que realizan procesos de investigación y pruebas.

Nevada fue el primer estado en promulgar este tipo de leyes, que permiten la conducción autónoma. En este caso, concedió la primera licencia a un vehículo sin conductor cuya tecnología, en fase experimental, pertenecía al desarrollador *Google*.

- **Europa:**

En Burdeos, los primeros vehículos autónomos comenzaron a integrarse en el flujo del tráfico abierto durante el año 2015, coincidiendo con el Congreso Mundial del Transporte Inteligente. En ese congreso, se presentaron vehículos implementados por cinco empresas. Estos vehículos son identificables gracias a un registro específico que advierte a los usuarios de su presencia.

Sin embargo, los responsables y organizadores del congreso tuvieron que levantar ciertas restricciones instauradas. Incluso, en Reino Unido se afirmó durante el congreso que no existían barreras legales en ese momento para la circulación de vehículos con estas características.

A pesar de esa afirmación, es conocida la necesidad de estudio y modificación de las normas de control vehicular para permitir que los vehículos autónomos circulen de manera legal.

Por otro lado, en Suecia se han realizado pruebas de camiones no tripulados en la zona minera de Boliden, y durante el año 2017 confirmaban el permiso a mil vehículos Volvo con la tecnología autónoma integrada para transitar por las calles de Gotemburgo.

- **España:**

En 2016 se afirmaba en España la inminente llegada de la revisión del código regulador que permitiría la inclusión de estos vehículos en las calles impulsada por la llegada del fabricante Tesla.

El gobierno trabaja en un plan llamado Movilidad Autónoma y Conectada, en cuyo reglamento se pretendía aprobar durante el año 2017 los primeros cambios con respecto a la conducción diaria. La idea en España consiste en la aplicación de una legislación flexible y bien definida que ofrezca seguridad jurídica adaptada a la actualidad.

El responsable de la DGT, Arriola, afirmaba el año pasado lo siguiente: "trabajamos en paralelo para modificar la ley de seguro obligatorio y la de seguridad vial". También comentó, que la aprobación de dicha modificación se retrasaría a este año 2018 o incluso al 2019, debido a la necesidad de colaboración de varios Ministerios para cada una de las leyes y debido también a la tramitación parlamentaria.

Las últimas noticias sobre este tema, que recogía el diario español "El País" sobre este tema, comentaba que la Comisión Económica de las Naciones Unidas para Europa, la UNECE, ha introducido algunas enmiendas en la Convención de Viena sobre el tráfico rodado perteneciente al año 1968, la cual establece la normativa en carretera a nivel internacional, con el fin de permitir las tecnologías de conducción automatizada.

La Fiscalía especializada en Seguridad Vial se plantea si habrá un "desplazamiento de las responsabilidades de este orden desde el conductor hacia el fabricante del vehículo bajo el título de responsabilidad objetiva o principio del riesgo o con el mismo fundamento hacia los fabricantes del software que gestiona el sistema". Y extiende esta reflexión hacia la industria que lo explota, no necesariamente coincidente con el fabricante, e incluso hacia los elaboradores de los mapas cartográficos utilizados por el sistema.

La Fiscalía señala que "también habrá que tener en cuenta la responsabilidad patrimonial de la Administración Pública cuando los vehículos autónomos circulen prestando servicios públicos". El Ministerio Público señala en ese artículo, además, que "incluso en los casos de plena automatización podría plantearse la responsabilidad penal del eventual conductor cuando el vehículo estuviera circulando en ese momento con la tecnología autónoma desactivada, posibilidad que el sistema ofrece, y el siniestro se deba a la conducta imprudente por omitir elementales deberes, como ya sucede en la actualidad con los vehículos convencionales".

Y añade para concluir que, en la investigación y depuración de responsabilidades, podría ser de gran utilidad la instalación de "cajas negras" en estos vehículos inteligentes (*Event Data Recorder*) que informen de si en el momento del siniestro conducía el sistema informático o lo hacía el conductor físico y refleje los eventuales fallos o disfunciones del

sistema, por lo que habrá que explorar la regulación de esta materia y sus repercusiones en la privacidad o protección de datos.

7.2. Legislación del desarrollo Software

En el caso particular del proyecto aquí expuesto, al tratarse del sistema operativo *Ubuntu Linux*, sistema de desarrollo de Software libre, no está citado en ninguna legislación. Tras la lectura de la *GPL* (*General Public License*), lo único aplicable a la legislación vigente que afecta a este sistema tiene que ver con el concepto mercantil de franquicia [30].

La similitud existe: existe una marca comercial: *Ubuntu*; un propietario: el creador o dueño del *copyright*; que establecen unos derechos de uso y explotación de la marca comercial: la licencia *GPL*.

Las diferencias existentes son especialmente notables en cuanto al uso del derecho de explotación por parte de terceras personas (redistribución). En cualquier caso, se hace patente una necesidad fundamental en el ámbito del Software libre, que consiste en la garantía de titularidad del producto. Este punto es el único aplicable a los entornos de desarrollo basados en el Software libre: la titularidad.

Es fundamental que la titularidad sea reconocida y mantenida bajo toda circunstancia para que el modelo de Software libre sea posible. Es necesario el registro del programa, y la licencia otorgada debe reflejarlo claramente.

Las leyes de la Unión Europea precisan de un contrato o documento de aceptación explícita respecto a la licencia, con el conocimiento mutuo. Se admite como firma del contrato, la solicitud de clave de registro o la introducción de dicha clave en el proceso de instalación o activación de un programa determinado.

Por otro lado, de cara a la licencia *GPL*, implica que su aceptación debe ser de la misma manera. En este caso, no se considera válida la instalación o acceso a una determinada aplicación para su uso como validación de dicha licencia. Sino, que es necesario el registro para que dicha licencia tenga vigencia en el continente.

Por ello, es necesario que el Software libre esté perfectamente registrado en el *copyright* vigente.

8. ENTORNO SOCIOECONÓMICO

En este apartado se desarrolla el impacto que el vehículo autónomo y el ámbito que engloba están causando sobre la sociedad actual y los principales mercados.

8.1. Impacto Social

Este apartado enmarca el impacto que de manera casi segura producirá la llegada del vehículo inteligente a la sociedad actual. Como se anticipaba en el capítulo 1. de este documento de manera introductoria, la llegada del vehículo inteligente es inminente, más de lo que se preveía unos años antes.

La seguridad y la comodidad en la conducción son los principales logros que se obtendrían de la llegada de la conducción inteligente de vehículos, y esto, además, eliminaría el factor humano que tantos accidentes causa en la sociedad.

No obstante, también se ha explicado en otros capítulos, y apoyándose del artículo de Joel Janai, Fatma Güney, Asheem Behl, y Andreas Geiger, la problemática actual de los sistemas de percepción que captan la información para el vehículo, la cual muestra aún errores que no pueden ser aceptados para la conducción en espacios abiertos no controlados.

Por último, también se ha explicado en el capítulo anterior la legislación actual, y los cambios que se quieren realizar para adaptar el vehículo a la sociedad, aunque aún tardará un tiempo en llevarse a cabo.

Pero, es necesario centrarse también en cómo afecta positiva y negativamente en la sociedad la incursión de este vehículo futurista, como explicaba el periodista Alejandro Nieto González hace 5 años.

La llegada de un vehículo autónomo, que no necesita de capacidades humanas para el transporte, podría afectar a muchas profesiones del sector de los transportes [30]. Las profesiones en riesgo serían las de chófer, taxista, conductor de autobuses, camionero, mensajeros que trabajan en moto o furgoneta, transportistas de todo tipo de sectores, etc. Esto es, las profesiones relacionadas con la carretera correrían grave peligro, pues la tecnología del vehículo autónomo pretende aportar mayor seguridad a la conducción con respecto a la conducción humana.

Otra posibilidad a tener en cuenta pueden ser las autoescuelas, como comenta el editor, ya que, cabe la posibilidad el plantearse la necesidad de tener o no un profesor durante las clases de conducir.

Todos estos argumentos son diferentes puntos que deben ser tenidos en cuenta para que la llegada del vehículo autónomo no cause problemas en la sociedad, pero, por otro lado, sería necesario también qué persona pasa a ser la responsable directa de un vehículo autónomo que, por problemas eléctricos, electrónicos o mecánicos, ocasionase un accidente, un impacto contra algún obstáculo, o un atropello.

8.2. Principales cambios en los mercados

Un estudio realizado en el año 2017 para *Intel*, por parte de la empresa *Strategy Analytics* llamado "Acelerando el futuro: el impacto económico en la emergente economía de pasajeros", determinaba el impacto ocasionado sobre los accidentes anuales.

Intel afirma, que, teniendo en cuenta que se producen al año hasta 1.3 millones muertes a causa de los accidentes de tráfico, y que, de estas muertes, 1.2 millones se deben a errores humanos, la introducción del vehículo autónomo podría salvar hasta casi 600.000 vidas, si mejorase este aspecto en tan solo un 5% [31].

Por otro lado, en el artículo publicado por la editora *Europa Press*, explica que la OMS (Organización Mundial de la Salud) estima que "en Estados Unidos se ahorrarían 234.000 millones de dólares con la implantación de los vehículos si logran reducir un 1% anual los costes sanitarios relacionados con los accidentes de tráfico entre 2035 y 2045" basándose en un gasto medio del 3% del PIB (Producto Interior Bruto) por parte de los gobiernos [32].

La empresa que realizó el estudio, *Strategy Analytics*, estima que la cumbre de este sector llegará en dos o tres décadas, hacia el año 2040, y concluyó en su informe, que el uso del vehículo autónomo como parte del sector de la movilidad podrá generar ganancias cercanas a los 3 billones de dólares, cifra que se traduce actualmente al 43% del total de la economía de pasajeros.

Concluye, también incluyendo en dicho informe, una ganancia de hasta 3.7 billones de dólares, el 55% de la economía de pasajeros a través de posibles ofertas transporte como servicio.

9. PRESUPUESTO

En este apartado, se explica el método de obtención del presupuesto necesario para la realización y desarrollo del proyecto descrito en este documento, así como el desglose de los costes y presupuesto total mostrados en las figuras mostradas al final de este apartado.

Este presupuesto se obtiene mediante el desglose de dos campos principales:

- Costes de material:

Para la obtención de los costes producidos por el material empleado, es necesario incluir el ordenador portátil utilizado para el desarrollo del código del visualizador y la aplicación *Android*.

Además, ha sido necesaria la utilización de un dispositivo móvil con sistema operativo *Android* cuya versión de Software fuese superior a la versión 4.0 *Icecream Sandwich.*

Por último, se ha utilizado también el sistema operativo *Linux* proporcionado por *Ubuntu* para la utilización de la arquitectura de control robótico ROS.

- Cálculo de costes de material:

Los costes materiales se han calculado en base al valor de los materiales utilizados, el tiempo empleado en la consecución de los resultados, y el coeficiente de amortización proporcionado por la tabla de amortización simplificada del Ministerio de Hacienda [33], mostrada a continuación:

TABLA 9.1.
TABLA DE AMORTIZACIÓN SIMPLIFICADA DE LA AGENCIA TRIBUTARIA

Grupo	Elementos patrimoniales	Coeficiente lineal máximo (%)	Período Máximo (Años)
1	Edificios y otras construcciones	3	68
2	Instalaciones, mobiliario, enseres y resto del inmovilizado material	10	20
3	Maquinaria	12	18
4	Elementos de Transporte	16	14
5	Equipos para tratamiento de la información y sistemas y programas informáticos	26	10
6	Útiles y herramientas	30	8
7	Ganado vacuno, porcino, ovino y caprino	16	14
8	Ganado equino y frutales no cítricos	8	25
9	Frutales cítricos y viñedos	4	50
10	Olivar	2	100

"Tabla de amortización simplificada". (Agencia Tributaria (AEAT), 2018) [Ref]

Por tanto, el coste se calcula según la siguiente fórmula:

$$Costes\ de\ material = precio\ x\ \frac{coef.amortización}{100}\ x\ \frac{tiempo\ (meses)}{12\ meses} \quad (9.1)$$

- Costes de recursos humanos:

Los costes referidos a los recursos humanos (RRHH) utilizados se calculan a partir del sueldo medio obtenido por un ingeniero *junior* en España actualmente: entre los 26.000 y los 30.000 € brutos al año.

- Cálculo de costes de recursos humanos:

El cálculo total del coste de recursos humanos se realiza a partir de la siguiente fórmula:

$$Costes\ de\ RRHH = Sueldo\ medio\ \left({}^{€}/_{mes}\right) x\ \frac{{}^{nº\ horas}/_{semana}}{40\ {}^{horas}/_{semana}}\ x\ nº\ meses \qquad (9.2)$$

Es necesario destacar que, en el caso de este proyecto, el sueldo mensual debe considerarse en base a una jornada de 25 horas semanales, es decir, media jornada aproximadamente.

TABLA 9.2.
COSTES MATERIALES GENERADOS

DESGLOSE DE COSTES DE MATERIALES					
CONCEPTO	DESCRIPCIÓN	COEFICIENTE DE AMORTIZACIÓN	COSTE UNITARIO	TIEMPO EMPLEADO	COSTE FINAL
PC PORTÁTIL PERSONAL	HP NOTEBOOK 15" i3 8GB RAM 500GB HDD	26%	529.99€	17 meses	187.70€
SMARTPHONE PERSONAL	SAMSUNG GALAXY S6 4GB RAM 32 GB HDD	26%	819.99€	17 meses	302.03€
SISTEMA OPERATIVO	UBUNTU LINUX	26%	0 €	18 meses	0 €

TABLA 9.3.
COSTES DE RECURSOS HUMANOS

DESGLOSE DE COSTES DE RECURSOS HUMANOS				
PUESTO DE TRABAJO	SUELDO MENSUAL	JORNADA	TIEMPO EMPLEADO	COSTE FINAL
Ingenierio Junior	1.800 €	60%	17 meses	18.360 €

TABLA 9.4.
PRESUPUESTO FINAL

PRESUPUESTO TOTAL
18.849'73€

10.CONCLUSIONES

En este capítulo se exponen las conclusiones del proyecto, justificando la consecución de los objetivos propuestos y su funcionalidad. En primer lugar, es necesario resumir los objetivos buscados durante la ejecución del proyecto:

- Por un lado, se buscaba la obtención de un visualizador de nubes de puntos funcional y práctico, con el fin de comprobar de manera rápida y cómoda la correcta funcionalidad de los sistemas de detección implementados en el vehículo autónomo.

- Y, en segundo lugar, se buscaba la implementación de una aplicación con SO *Android* con el objetivo de: mejorar la calidad de trabajo de los compañeros del departamento encargado de la investigación del vehículo autónomo, obtener de manera sencilla y rápida información de los entornos detectados durante la conducción del vehículo, y aumentar la comodidad de poder visualizarlo de manera rápida en un dispositivo portátil.

- Por último, las dos implementaciones tienen un objetivo común: mejorar la confianza de los pasajeros del vehículo respecto al vehículo.

Por tanto, se puede comprobar que los objetivos buscados y descritos al comienzo de este documento han sido satisfechos.

El visualizador de nubes de puntos implementado muestra de manera rápida y óptima los datos publicados en los *topics* de nubes de puntos. Añadiendo alguna o varias de las mejoras propuestas en el capítulo 6., se obtendría un visualizador más completo y con mayor funcionalidad, a pesar de cumplir las expectativas en su estado actual.

Por otro lado, la aplicación de *Android* también cumple con las expectativas, ya que, es capaz de conectarse a un PC de manera remota, acceder a la información publicada por los nodos de ROS en ejecución en el ordenador, y filtrarla según el tipo de dato requerido, mostrando los resultados en una lista. Además, muestra las imágenes captadas por las cámaras en color, en blanco y negro, y también muestra las nubes de puntos publicadas.

Cabe destacar, que la aplicación *LSI Play Remote* obtenida se puede considerar un gran avance, dado que actualmente, no se conocen aplicaciones con una funcionalidad parecida, capaz de actuar de manera correcta entre diferentes pantallas, sin perder la conexión establecida de manera remota con el PC. Al contrario que lo logrado en esta aplicación, los desarrollos existentes hasta la fecha no funcionan con cambios de pantalla, sino que tienen funcionalidades más simples definidos en una única pantalla.

Además, el modelo propuesto ha sido obtenido después de lidiar con bastantes problemas referidos a las versiones de *Android*, *Java*, y ROS. Por ejemplo, en la última actualización de versiones del código soportado por el SO *Android*, se eliminaron métodos que habrían facilitado el trabajo con esta aplicación.

Una vez se apliquen algunas de las operaciones descritas en los posibles trabajos futuros, la aplicación tendrá la funcionalidad suficiente como para ser integrada en el vehículo autónomo.

Por último, se puede concluir, que los objetivos han sido alcanzados de manera satisfactoria, puesto que ambas implementaciones serán utilizadas en el trabajo diario de investigación de los compañeros del Laboratorio de Sistemas Inteligentes de la Universidad Carlos III de Madrid.

11.BIBLIOGRAFÍA

[1] Xavier Alegret, "¿Cuántos coches circulan por el mundo?". Economía Digital (2016). Recuperado de: https://www.economiadigital.es/politica-y-sociedad/cuantos-coches-circulan-por-el-mundo_182150_102.html
[Último acceso: junio de 2018]

[2] RTVE, "Presupuestos Generales del Estado 2018". RTVE.es / EFE (abril de 2018). Recuperado de: http://www.rtve.es/noticias/20180403/presupuestos-destinan-6366-millones-para-ciencia-este-ano-54-mas-2017/1708004.shtml
[Último acceso: junio de 2018]

[3] ABC, "Pasamos 9 horas y 35 minutos de media semanales metidos en el coche". ABC Motor (abril de 2017). Recuperado de: http://www.abc.es/motor/reportajes/abci-pasamos-9-horas-y-35-minutos-media-semanales-metidos-coche-201704271427_noticia.htm
[Último acceso: junio de 2018]

[4] "Tablas estadísticas". Web de la Dirección General de Tráfico (junio de 2018). Recuperado de: http://www.dgt.es/es/seguridad-vial/estadisticas-e-indicadores/accidentes-30dias/tablas-estadisticas/
[Último acceso: junio de 2018]

[5] Benjamín, "Ingeniería de Sistemas". Web *Wikipedia* (mayo de 2018). Recuperado de: https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_sistemas
[Último acceso: junio de 2018]

[6] A. Vicente Medina, L. Alberto Galarza, "Vehículos Autónomos". (febrero de 2016). Recuperado de: http://www.monografias.com/trabajos109/estado-del-arte-vehiculos-autonomos/estado-del-arte-vehiculos-autonomos.shtml
[Último acceso: junio de 2018]

[7] C. Sánchez, "Ernst Dickmanns, el desconocido padre alemán de los coches inteligentes". El Diario.es (abril de 2015). Recuperado de: https://www.eldiario.es/hojaderouter/tecnologia/Ernst_Dickmanns-vehiculo-autonomo-inteligente_0_382511814.html
[Último acceso: junio de 2018]

[8] I. Palou, "NavLab, un coche autónomo de 1986". Economía Digital (noviembre de 2016). Recuperado de: http://www.microsiervos.com/archivo/coches/navlab-coche-autonomo-1986.html
[Último acceso: junio de 2018]

[9] J. Janai, F. Gütney, A. Behl, A. Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-art". arXiv: 1704.05519vl [cs.CV], (abril de 2017).

[10] J. Manuel Blanco, "Diez años de la mítica competición que encendió el motor del coche autónomo". El Diario.es (noviembre de 2017). Recuperado de: https://www.eldiario.es/hojaderouter/movilidad/mitica-competicion-encendio-motor-autonomo_0_705179683.html

[Último acceso: junio de 2018]

[11] A. Geiger, P. Lenz, R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". Conference on Computer Vision and Pattern Recognition (CVRP), (2009).

[12] Equipo de marketing, "¿En qué consiste el RTLS?" Wellness Telecom (julio de 2017). Recuperado de: http://www.wtelecom.es/2017/07/real-time-location-system/
[Último acceso: junio de 2018]

[13] J. Valero, "El futuro de los sensores LIDAR para coches autónomos". Hipertextual (Sección Ford) (enero de 2016). Recuperado de: https://hipertextual.com/presentado-por/ford/sensores-coches-autonomos
[Último acceso: junio de 2018]

[14] P. Ibañez, "¿Qué es un LIDAR y cómo funciona el sensor más caro de los coches autónomos?". Motorpasion (septiembre de 2017). Recuperado de: https://www.motorpasion.com/tecnologia/que-es-un-lidar-y-como-funciona-el-sistema-de-medicion-y-deteccion-de-objetos-mediante-laser
[Último acceso: junio de 2018]

[15] ISLab, "Advanced Driver Assistance Systems", Universidad Carlos III de Madrid, Laboratorio de Sistemas Inteligentes (2016). Recuperado de: http://portal.uc3m.es/portal/page/portal/dpto_ing_sistemas_automatica/investigacion/IntelligentSystemsLab/research/adas#IVVI2
[Último acceso: junio de 2018]

[16] Internet Archive, "Kinect". Web Wikipedia (marzo de 2018). Recuperado de: https://es.wikipedia.org/wiki/Kinect
[Último acceso: junio de 2018]

[17] Android Web, "Android Auto: La mejor información para la carretera". Android (2017). Recuperado de: https://www.android.com/intl/es_es/auto/
[Último acceso: junio de 2018]

[18] I. Teso, "Android Auto: ¿Qué es y cómo funciona? Coches.com (diciembre de 2016). Recuperado de: https://noticias.coches.com/consejos/android-auto-como-funciona/233264
[Último acceso: junio de 2018]

[19] M. Gurman, A. Webb, "How Apple scaled back its titanic plan take on Detroit". Bloomberg (octubre de 2016). Recuperado de: https://www.bloomberg.com/news/articles/2016-10-17/how-apple-scaled-back-its-titanic-plan-to-take-on-detroit
[Último acceso: junio de 2018]

[20] Alberto, "iCar, ¿en qué punto está el coche autónomo de Apple? Nobbot (enero de 2018). Recuperado de: https://www.nobbot.com/futuro/icar-coche-autonomo/
[Último acceso: junio de 2018]

[21] Documento web, "Ubuntu". Web Wikipedia (junio de 2018). Recuperado de: https://es.wikipedia.org/wiki/Ubuntu
[Último acceso: junio de 2018]

[22] Documento web, "Sistema Operativo Robótico". Web Wikipedia (diciembre de 2017). Recuperado de: https://es.wikipedia.org/wiki/Sistema_Operativo_Rob%C3%B3tico
[Último acceso: junio de 2018]

[23] I. Recio, "ROS Wiki". Web oficial ROS.org (febrero de 2018). Recuperado de: http://wiki.ros.org/es
[Último acceso: junio de 2018]

[24] Documento web, "Android Studio". Web Wikipedia (mayo de 2018). Recuperado de: https://es.wikipedia.org/wiki/Android_Studio
[Último acceso: junio de 2018]

[25] J. Olano, "Git". Web Wikipedia (junio de 2018). Recuperado de: https://es.wikipedia.org/wiki/Android_Studio
[Último acceso: junio de 2018]

[26] R. Dudler "Git – La guía sencilla". Github (s.f.). Recuperado de: http://rogerdudler.github.io/git-guide/index.es.html
[Último acceso: junio de 2018]

[27] Documento web, "OpenGL". Web Wikipedia (enero de 2018). Recuperado de: https://es.wikipedia.org/wiki/OpenGL
[Último acceso: junio de 2018]

[28] Documento web, "Fragmentos". Developers Android (abril de 2018). Recuperado de: https://developer.android.com/guide/components/fragments?hl=es-419
[Último acceso: junio de 2018]

[29] N. López, "Legislación sobre el coche autónomo en España: queda mucho por hacer". Revista Autobild Versión Digital (diciembre de 2017). Recuperado de: https://www.autobild.es/reportajes/legislacion-coche-autonomo-espana-queda-mucho-hacer-179660
[Último acceso: junio de 2018]

[30] Free Software Foundation, "Licencias". GNU Edición Digital (abril de 2018). Recuperado de: https://www.gnu.org/licenses/licenses.es.html
[Último acceso: junio de 2018]

[31] A. Nieto, "Coche autónomo y efectos económicos". El blog de Salmón (enero de 2017). Recuperado de: https://www.elblogsalmon.com/entorno/coche-autonomo-y-efectos-economicos
[Último acceso: junio de 2018]

[32] Vaznar, "Intel prevé que la conducción autónoma generará una nueva 'Economía de Pasajeros'". Globb IT (junio de 2017). Recuperado de: https://www.globbit.com/intel-

preve-la-conduccion-autonoma-generara-una-nueva-economia-pasajeros-valor-7b-10598/
[Último acceso: junio de 2018]

[33] Europa Press, "Los vehículos podrían salvar 585.000 vidas entre 2035 y 2045". Europa Press Motor Edición Digital (junio de 2017). Recuperado de: http://www.europapress.es/motor/coches-00640/noticia-vehiculos-autonomos-podrian-salvar-585000-vidas-2035-2045-20170602145554.html
[Último acceso: junio de 2018]

[34] Agencia Tributaria, "Tabla de amortización simplificada". Recuperado de: http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresarios_individuales_y_profesionales/Rendimientos_de_actividades_economicas_en_el_IRPF/Regimenes_para_determinar_el_rendimiento_de_las_actividades_economicas/Estimacion_Directa_Simplificada.shtml
[Último acceso: junio de 2018]